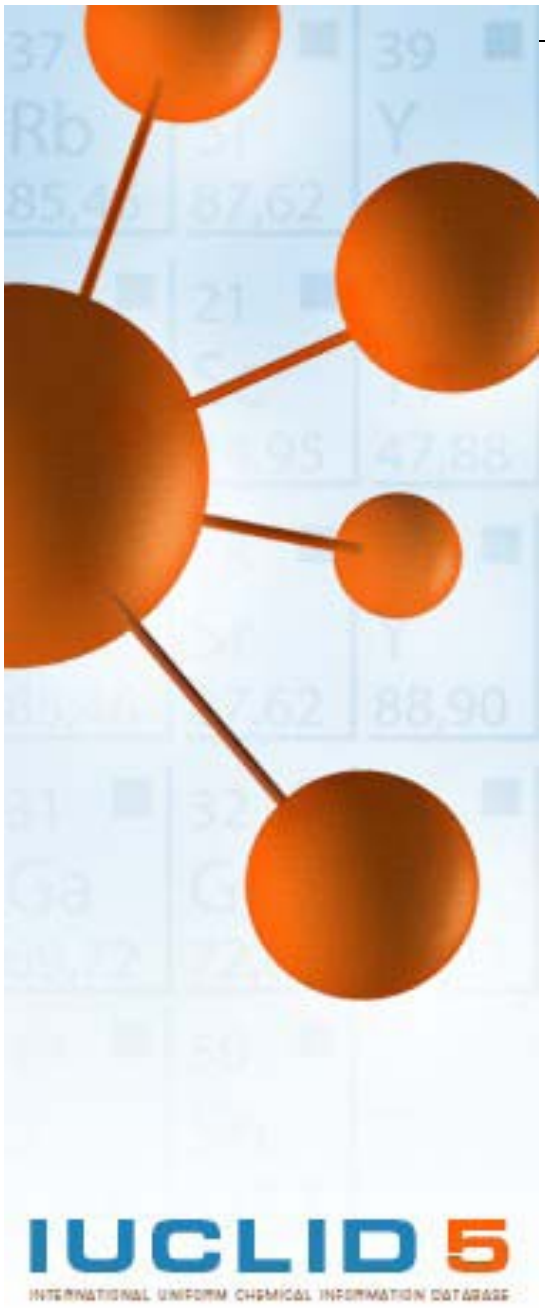

IUCLID 5
Guidance and Support

Web Service Technical Manual
(for developers)

May 2009
Version 1.0

IUCLID 5 has been developed by the European Commission in association with the OECD

Table of contents

1. INTRODUCTION 1

1.1. TECHNOLOGY 1
1.2. SAMPLE PROGRAMS 2

2. SESSION ENGINE 3

2.1. INTERFACE METHODS 3
2.1.1. Login 4
2.1.2. Logout 6

3. QUERY ENGINE 6

3.1. SHORT INTRODUCTION TO DATA MODEL 6
3.2. INTERFACE METHODS 9
3.2.1. getAllQueryNames 9
3.2.2. getAllQueryResultTypes 11
3.2.3. getAllQueryDefinitions 11
3.2.4. getQueryDefinitionByName 12
3.2.5. getQueryDefinitionsByName 12
3.2.6. getQueryDefinitionsByResultDocumentType 12
3.2.7. executeQuery 13
3.3. HOW TO USE THE QUERY ENGINE 14

4. DOCUMENT ACCESS ENGINE 15

4.1. SHORT INTRODUCTION TO DATA MODEL 16
4.2. INTERFACE METHODS 16
4.2.1. existsDocument 16
4.2.2. determineDocumentType 17
4.2.3. closeDocument 17
4.2.4. fetchDocument 18
4.2.5. openDocumentForReading 19
4.2.6. compareDocuments 20
4.2.7. openDocumentForWriting 21
4.2.8. commitDocument 22
4.2.9. deleteDocument 23
4.2.10. createDocument 24
4.3. HOW TO USE THE DOCUMENT ACCESS ENGINE 25

5. ENDPOINT ACCESS ENGINE 26

5.1. INTERFACE METHODS 26
5.1.1. createNewEndpoint 26
5.1.2. createNewEndpointSummary 27
5.1.3. copyEndpoint 28
5.1.4. copyEndpoints 29
5.1.5. deleteEndpoint 30
5.1.6. deleteEndpoints 30
5.1.7. referenceEndpoint 31
5.1.8. referenceEndpoints 32
5.1.9. updateEndpoint 33
5.1.10. detachEndpoint 34
5.2. HOW TO USE THE ENDPOINT ACCESS ENGINE 34

6. REFERENTIAL INTEGRITY ENGINE 35

6.1. INTERFACE METHODS 36
6.1.1. check 36
6.2. HOW TO USE THE REFERENTIAL INTEGRITY ENGINE 36

7. QUERY TOOL ENGINE 37

7.1. INTERFACE METHODS 37
7.1.1. countQuery 37
7.1.2. executeQuery 38
7.2. DATA STRUCTURES 39
7.2.1. Query Expression 39
7.2.2. Query Result 41
7.3. HOW TO USE THE QUERY TOOL ENGINE 41

8. INVENTORY ENGINE 45

8.1. INTERFACE METHODS 45
8.1.1. getLiteratureInventoryQuery 45
8.1.2. executeLiteratureInventoryQuery 47
8.1.3. putLiteratureInventoryEntry 47
8.1.4. putLiteratureInventoryEntries 48
8.1.5. deleteLiteratureInventoryEntry 48
8.1.6. deleteLiteratureInventoryEntries 48

9. MESSAGING ENGINE 49

9.1. INTERFACE METHODS 49
9.1.1. broadcastShowDocument 49

10. REFERENCE TRACKER ENGINE 50

10.1. INTERFACE METHODS 50
10.1.1. queryInbound 50
10.1.2. queryOutbound 51

11. RESOURCE ENGINE 52

11.1. SHORT INTRODUCTION TO DATA MODEL 53
11.2. INTERFACE METHODS 54
11.2.1. getAvailableLanguages 54
11.2.2. getResource 55

12. CONTAINER ENGINE 55

12.1. INTRODUCTION 55
12.1.1. Limitations 56
12.1.2. Technical Conditions 56
12.1.3. Transferring Binary Data 57
12.2. CONTAINER ENGINE 58
12.2.1. Short Introduction to Data Model 58
12.2.2. Interface Methods 59
12.2.2.1. uploadContainer 59
12.2.2.2. importExternalContainer 59
12.2.2.3. downloadContainer 60
12.3. ADDITIONAL CONFIGURATION 60

13. SOAP 1.1 BINDING 61

14. ANNEX 62

14.1. DOCUMENT IDENTIFIERS – DOCUMENTREFERENCEPKWS 62
14.2. RANDOM UUIDS 63
14.3. SPECIAL UUIDS 64

1. Introduction

The web service standard is the industry's choice for interfacing a software product with external systems. A web service provides platform-neutral and language-neutral interfaces that are self-describing (WSDL) and leverage a variety of supporting tools.

This documentation gives you a detailed description of the available web service endpoints and how you have to use them and interact with your client application.

This document **does not** explain the web service technology or other technical topics. It covers the usage of the IUCLID 5 web service interface and the dependencies between its different endpoints. In addition to this manual, a detailed understanding of the web service technology is needed.

IUCLID 5 is a complex Software System which is used to store and manage a large number of chemical data based on the OECD Harmonised Templates structure. A web service interface cannot reduce the complexity of the software.

1.1. Technology

To implement the IUCLID 5 web service interface the AXIS2 web service framework was used, which can be downloaded from the AXIS2 project website (http://ws.apache.org/axis2/).

The used SOAP version is 1.2 and the WSDL are defined by using the version 1.1. During the development phase it was realised that the tool support for WSDL 2.0 was not as good as initially expected which caused many problems in generating stable web service stubs.

1.2. Sample Programs

For each web service endpoint you will find one sample program which should be used as reference implementation. They are included under 'Sample Programs' in the Web Service installation package and implemented as JUnit test cases.

Web Service End point	Sample Program
Session Engine	eu.eca.iuclid.webservice.service.sessionengine.SessionEngineTest
Query Engine	eu.eca.iuclid.webservice.service.queryengine.QueryEngineTest
Document Access Engine	eu.eca.iuclid.webservice.service.documentaccessengine. DocumentAccessEngineTest
Endpoint Access Engine	eu.eca.iuclid.webservice.service.endpointaccessengine. EndpointAccessEngineTest
Inventory Engine	eu.eca.iuclid.webservice.service.inventoryengine.InventoryEngineTest
Query Tool Engine	eu.eca.iuclid.webservice.service.querytoolengine.QueryToolEngineTest
Messaging Engine	eu.eca.iuclid.webservice.service.messagingengine.MessagingEngineTest
Container Engine	eu.eca.iuclid.webservice.service.containerengine.ContainerEngineTest

Table 0-1 List of Sample Programs

2. Session Engine

2.1. Interface Methods

The session engine service is used to handle the user authentication and session management. Some web service interactions require the saving of the client state. For instance if you want to change an existing substance you have to obtain a read-write lock at the IUCLID 5 server. These session states are managed by the Session Engine.

This means that a web service interaction always starts with a SessionEngine.login and ends with a SessionEngine.logout.

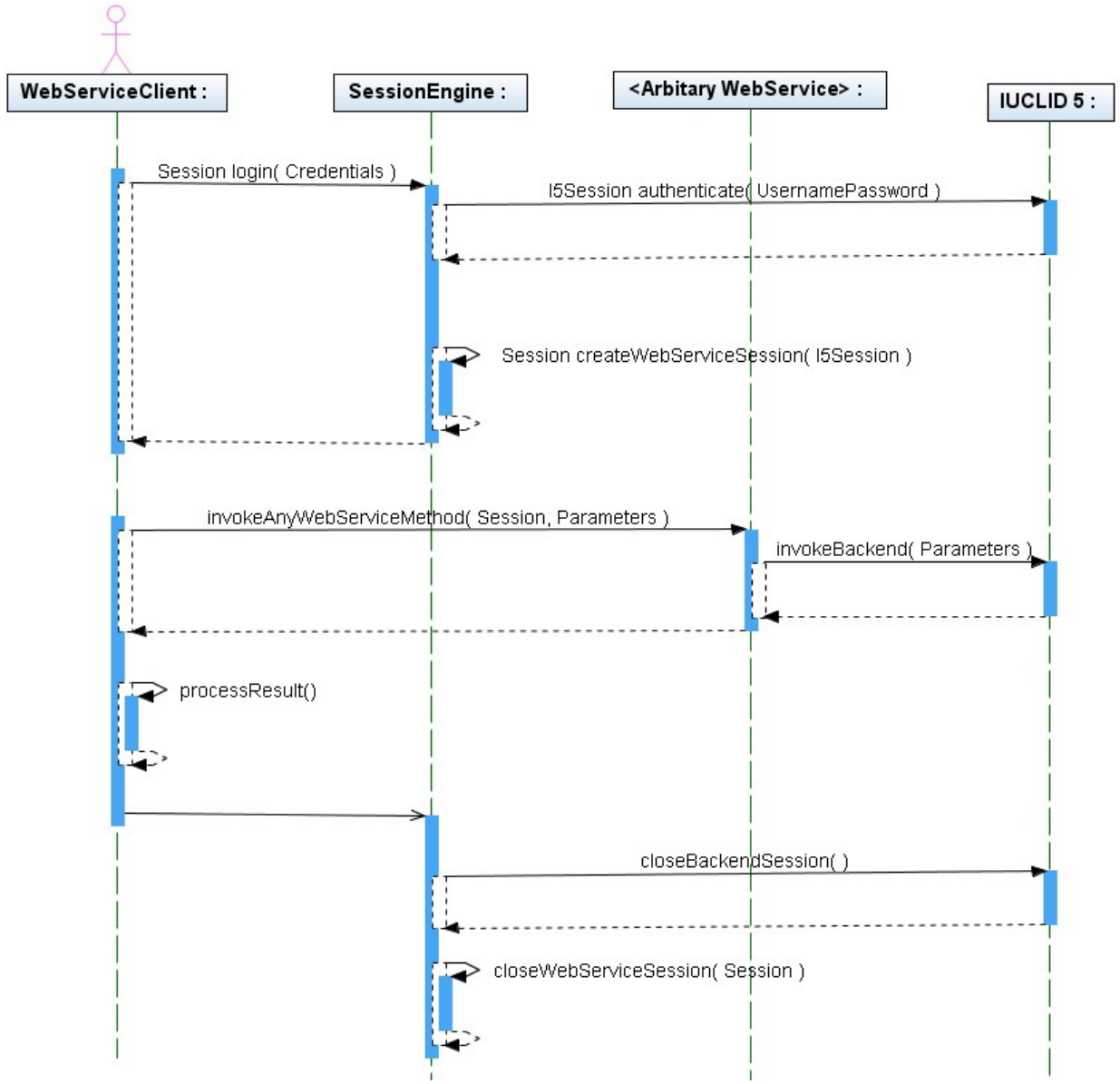

Figure 0-1 General Web Service Interaction

The web service session has a time-out period. If a session has not been used for the specified time, the user will be logged out automatically. In that case each non committed change will be lost. The default time out period is 7,200 seconds (=2 hours).

2.1.1. Login

Purpose	Obtain a web service session and login at the back-end system.
IN	• The users credential as <tns:Credentials> contains existing username and password. The password has to be sent as MD5 hash which was converted to its hex representation
OUT	• The new created Session contains the session token <tns:Session>
Prerequisites	You have an existing user at the connected IUCLID 5 server
Exceptions	• SessionEngineNotAvailableFault Is thrown when the back-end service is not available to fulfil the request • SessionEngineFault Is thrown for all occurred exception during the web service call

It is prohibited to send a password as clear text; IUCLID 5 will not accept it. The password has to be sent as MD5 hash which was converted to its hex representation.

The following code snipped shows one way of how to create such a hashed password. The Apache codec library was also tried, which created the correct password hash, except that the letters were lower case and **IUCLID** requires an **upper case** style.

```
...
/** Used building output as Hex */
private static final char[] DIGITS = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
                                       'A', 'B', 'C', 'D', 'E', 'F' };
...
...

/**
 * Encrypts the given password
 *
 * @param decryptedPassword
 *          the password
 * @return encrypted Password
 */
protected String encryptPassword( final String decryptedPassword )
{
   try
   {
      // Generates a java MessageDigest object that implements MD5
      // algorithm
      final MessageDigest md5 = MessageDigest.getInstance( "MD5" );

      // get the byte array of the password using UTF-8 encoding
      final byte[] pwdByte = decryptedPassword.getBytes( "UTF-8" );

      // create the MD5 for the given password byte array
      final byte[] md5Byte = md5.digest( pwdByte );
```

```
            // compute the hex representation
            return encodeHex( md5Byte );
        }
        catch( final NoSuchAlgorithmException ex )
        {
            throw new RuntimeException( ex );
        }
        catch( final UnsupportedEncodingException e )
        {
            // We catch this one because it would be a developer's error in the
            // hardcoded encoding string and should never happen.
            throw new RuntimeException( e );
        }
    }

    /**
     * Converts an array of bytes into an {@link String} of characters
     * representing the hexadecimal values of each byte in order.<br>
     * The returned array will be double the length of the passed array, as it
     * takes two characters to represent any given byte.
     *
     * @param hash
     *            Array of bytes to convert to hex-string
     * @return Generated hex string
     */
    public String encodeHex( final byte[] hash )
    {
        // get the current length
        final int currenLength = hash.length;

        // double the length of the output array
        final char[] out = new char[ currenLength * 2 ];

        // take always two characters form the hex value to get the double
        // length of the hashed byte array.
        for( int i = 0, j = 0; i < currenLength; i++ )
        {
            // calculate the used two hex values
            final int high = (int)(hash[ i ] >>> 4) & 0xf;
            final int low = (int)hash[ i ] & 0xf;
            out[ j++ ] = DIGITS[ high ];
            out[ j++ ] = DIGITS[ low ];
        }
        return new String( out );
    }
```

Code 0-1 Create MD5 Hex Password

2.1.2. Logout

Purpose	Close the IUCLID 5 session and release each held locks. Be careful if you have uncommitted changes, this call will not take care of them and the changes will be lost.
IN	• The open (logged in) session as <tns:Session>
OUT	This method do not define output parameters
Prerequisites	You have a valid and existing session token
Exceptions	• SessionEngineNotAvailableFault Is thrown when the back-end service is not available to fulfil the request • SessionEngineFault Is thrown for all occurred exception during the web service call

3. Query Engine

The Web Service exposes an interface to IUCLID standard queries. It is possible to ask the Web Service for the available query and its query definitions in order to fill up the correct query parameter and query ID-Name.

This Interface enables other IT-System to execute the standard IUCLID 5 queries which are accessible via the user interface.

3.1. Short introduction to data model

The query engine defines three important data structures:

- QueryDefinition. A query definition contains the meta data for one specific IUCLID 5 query and defines:
  - Resulting document type
    each query have only one define document type as result.
  - Query Name
    the name/ID to reference the query at the IUCLID 5 server
  - List of Query Parameter Definitions
    list of parameters which have an optional list of phrase values
    - List Phrase Values

The following diagram illustrates this structure:

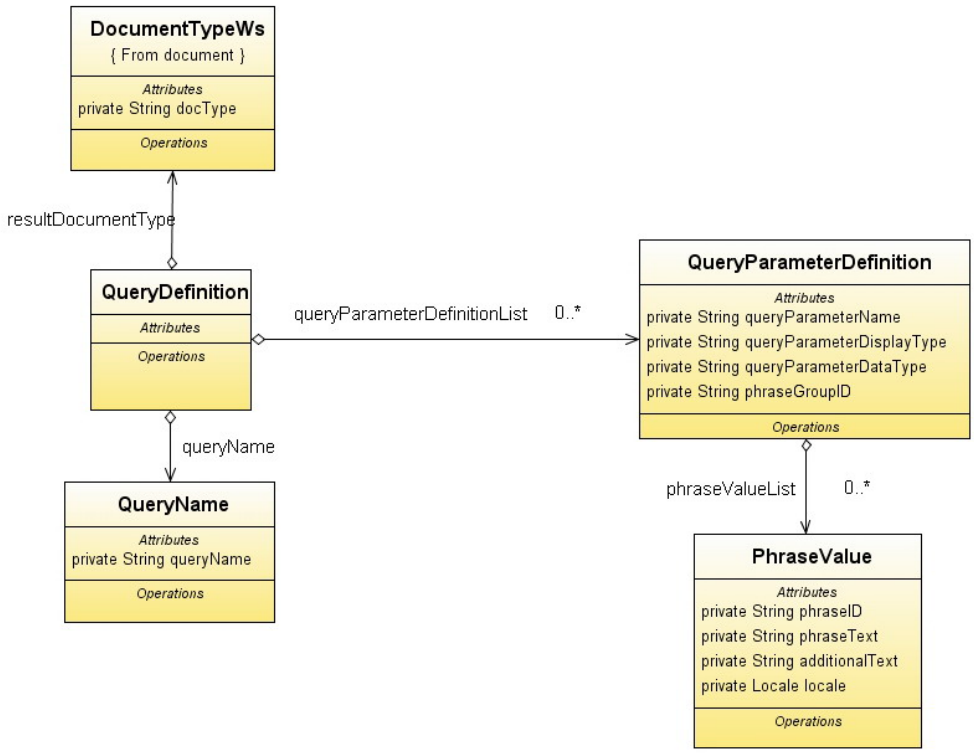

Figure 0-2 Query Definition Structure

Within the delivery you will find the document **QueryDefinitions.xls**, which contains a list of available queries and query parameters. The same list (as XML) would be returned by getAllQueryDefinitions.

- QueryExecutionDefiniton. Contains the meta data to execute a query:
  - Query Name, defined by its corresponding query definition
  - List of query parameters
    - Query parameter name, defined by its corresponding query definition
    - Query parameter value, the concrete parameter value to insert in the standard query. See Table 0-2 for details on the expected mapping between the specified query parameter display type and its query parameter value type.

Query Parameter Display Type	Query Parameter Value Type
DISPLAY_AS_CHECKBOX	BooleanParameterValue
DISPLAY_AS_DATE	DateParameterValue
DISPLAY_AS_DATE_RANGE	DateRangeParameterValue
DISPLAY_AS_INTEGER	IntegerParameterValue
DISPLAY_AS_PICKLIST	PicklistParameterValue
DISPLAY_AS_UUID DISPLAY_AS_STRING	TextParameterValue

Table 0-2 Mapping Query Parameter Display Type – Query Parameter Value Type

The following diagram illustrates this structure:

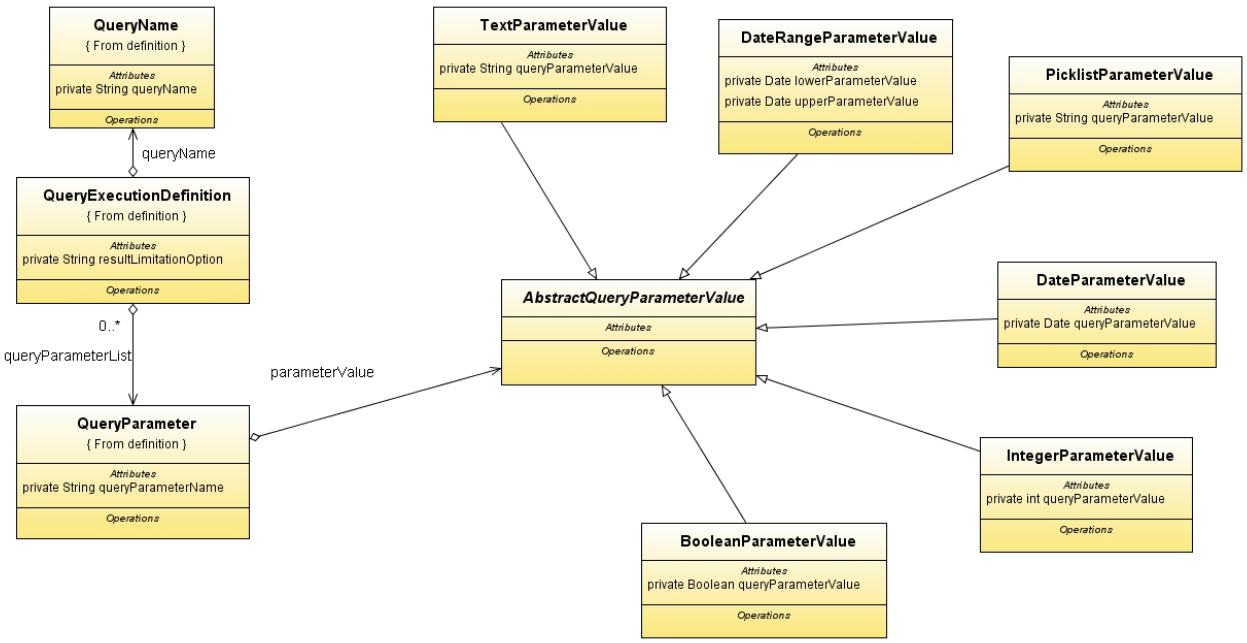

Figure 0-3 Query Execution Definition Structure

- QueryResultList. Contains the list of Document References.
  - List of document references
    - Document type, type of the target document.
    - Last modification date, last modification date of the target document.
    - Unique Key, unique identifier of the target document (contains the DocumentReferencePK).
    - Description, the alphanumeric description field which is displayed on screen as indicator where the document reference points to.
    - Notes, a free-text field for user convenience to attach additional text.

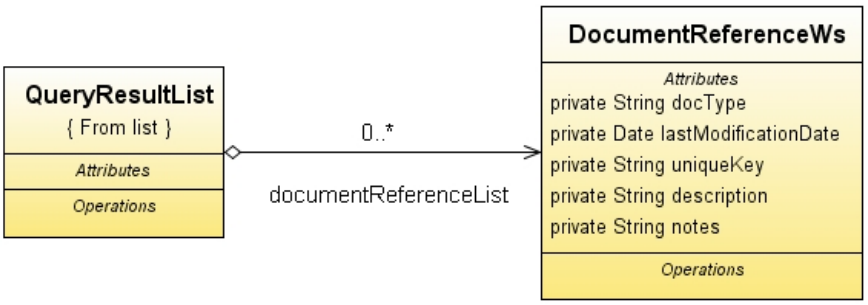

Figure 0-4 Query Result List Structure

3.2. Interface Methods

3.2.1. getAllQueryNames

Purpose	Returns all available query names. Each query could be identified by its query name (e.g. find_complex_substance). Before you can execute a query you need the Query Definition which defines the meta data of a query. To get this Query Definition you could use its query name.
IN	• The open (logged in) session as <tns:Session>
OUT	• A List (<tns:queryNameList>) of query names e.g. <tns:queryName id="find_complex_substance"/>
Exceptions	• QueryEngineNotAvailableFault Is thrown when the back-end service is not available to fulfil the request • QueryEngineFault Is thrown for all occurred exception during the web service call.

The following table shows the available query name IDs:

- **Substance**

find_all_substances	find_complex_substance_by_jointsubmission
find_complex_substance	find_complex_substance_by_additive
find_complex_substance_by_impurity	find_complex_substance_by_referencesubstance
find_complex_substance_by_legalentity	find_complex_substance_by_site
find_complex_substance_by_substancecomposition_constituent	
find_complex_substance_by_classification_and_labelling	

Table 0-3 Substance Query IDs

- **Mixture**

find_all_mixtures	find_complex_mixtures_by_constituent
find_complex_mixture	find_complex_mixtures_by_additive
find_complex_mixtures_by_impurity	find_complex_mixtures_by_related_substance

Table 0-4 Mixture Query IDs

- **Site**

find_all_sites	find_complex_site

Table 0-5 Site Query IDs

- **Category**

find_all_categories	find_complex_categories_by_legalentity
find_complex_categories	

Table 0-6 Category Query IDs

- **Template**

find_all_templates	find_complex_template

Table 0-7 Template Query IDs

- **Reference Substance**

find_all_referencesubstances	find_complex_referencesubstance

Table 0-8 Reference Substance Query IDs

- **Legal Entity**

find_all_legalentities	find_complex_legalentity

Table 0-9 Legal Entity Query IDs

- **Unknown**

find_complex_document_by_uuid

Table 0-10 Unknown Document Type Query IDs

- **Dossier**

find_all_dossiers	find_dossier_by_legal_entity
find_complex_dossier	find_dossier_by_reference_substance
find_dossier_by_substance	find_dossier_by_constituents
find_dossier_by_impurity	find_dossier_by_site
find_dossier_by_additive	find_dossier_by_classification_and_labelling
find_dossier_by_joint_submission	find_dossier_by_mixture

Table 0-11 Dossier Query IDs

3.2.2. getAllQueryResultTypes

Purpose	Returns a list of available and supported Document Result Types. IUCLID 5 defines several document types e.g. Substance, Category, Mixture (see http://iuclid.eu/index.php?fuseaction=home.format&type=public). Each IUCLID 5 Query defines one Document Result Type.
IN	• The open (logged in) session as <tns:Session>
OUT	• A List (<tns:queryResultTypeList>) of avalable document types e.g. <tns:documentType docType="Substance"/>
Exceptions	• QueryEngineNotAvailableFault Is thrown when e.g. the back-end service is not available to fulfil the request • QueryEngineFault Is thrown for all occurred exception during the web service call

The following table shows the available query result types:

Substance	Mixture
Site	Category
Template	ReferenceSubstance
Dossier	Unknown
LegalEntity	

Table 0-12 Query Result Types

3.2.3. getAllQueryDefinitions

Purpose	Returns **all** available Query Definitions (this call might take some time) A Query Definition contains the meta data of an IUCLID 5 query like name, Document Result Type and the list of Query Parameter Definitions. A Query Parameter Definition contains the name of the query parameter, the data type, the display type, the IUCLID 5 phrase group id and the corresponding list of phrases.
IN	• The open (logged in) session as <tns:Session>
OUT	• A List (<tns:queryDefinitionList>) of all available query definitions.
Exceptions	• QueryEngineNotAvailableFault Is thrown when e.g. the back-end service is not available to fulfil the request • QueryEngineFault Is thrown if an internal error occurred

3.2.4. getQueryDefinitionByName

Purpose	Return the specified Query Definitions by its name.
IN	• The open (logged in) session as <tns:Session> • The requested query name <tns:queryName id="find_all_legalentities"/>
OUT	• The found query Query Definition (see Figure 0-2 Query Definition Structure) <tns:queryDefinition />
Exceptions	• QueryEngineNotAvailableFault Is thrown when e.g. the back-end service is not available to fulfil the request • QueryEngineFault Is thrown if an internal error occurred

3.2.5. getQueryDefinitionsByName

Purpose	Return the requested list of Query Definitions by the requested list of query. If you need to load more than one query definition which has different Document Result Type you could use this method. It takes a list of Query Names and return the corresponding List of Query Definitions.
IN	• The open (logged in) session as <tns:Session> • A list (<tns:queryNameList>) of query names.
OUT	• A list (<tns:queryDefinitionList>) of Query Definitions
Exceptions	• QueryEngineNotAvailableFault Is thrown when e.g. the back-end service is not available to fulfil the request • QueryEngineFault Is thrown if an internal error occurred

3.2.6. getQueryDefinitionsByResultDocumentType

Purpose	Returns all Query Definitions which fit to the given Document Result Type.
IN	• The open (logged in) session as <tns:Session> • The requested DocumentTypeWs <tns:documentType docType="LegalEntity"/>
OUT	• A list (<tns:queryDefinitionList>) of Query Definitions which defines the requested DocumentType (<tns:documentType docType="LegalEntity"/>) as result document.
Exceptions	• QueryEngineNotAvailableFault Is thrown when e.g. the back-end service is not available to fulfil the request • QueryEngineFault Is thrown if an internal error occurred

3.2.7. executeQuery

Purpose	Execute the given query execution definition (see Figure 0-3 Query Execution Definition Structure).

IN	• The open (logged in) session as <tns:Session>
	• The Query Execution Definition which defines
	  o Query Name
	  o Query Parameter List
	  ▪ Query parameter name
	  ▪ Query parameter value

	• A list (<tns:queryResultList>) of Document References containing
	  o Document Type
	  o Last Modification Date
	  o Unique Key
	  o Description
	  o Notes

Exceptions	• QueryEngineNotAvailableFault Is thrown when e.g. the back-end service is not available to fulfil the request • QueryEngineFault Is thrown if an internal error occurred • QueryDeliveredTooManyHitsException This indicates a recoverable error while executing a query. E.g. this could be a too huge number of hits or errors resulting from other parameters the user was entered. • QueryTimeoutException The query reached its timeout limit before it completed. Query timeouts are generally enforced to prevent long running query to completely stall the system. This cannot fix the real culprit - the slow query, but is kind of a 'parachute' to keep the system operational. • MissingQueryParameterException This exception is thrown if the list of parameters does not fit the recommended list from the QueryParameterDefinition. You do not have to fill in all defined parameters but you should never put an undefined parameter into the list.

3.3. How to use the Query Engine

The following UML Sequence Diagram demonstrates one possible usage of the Query Engine.

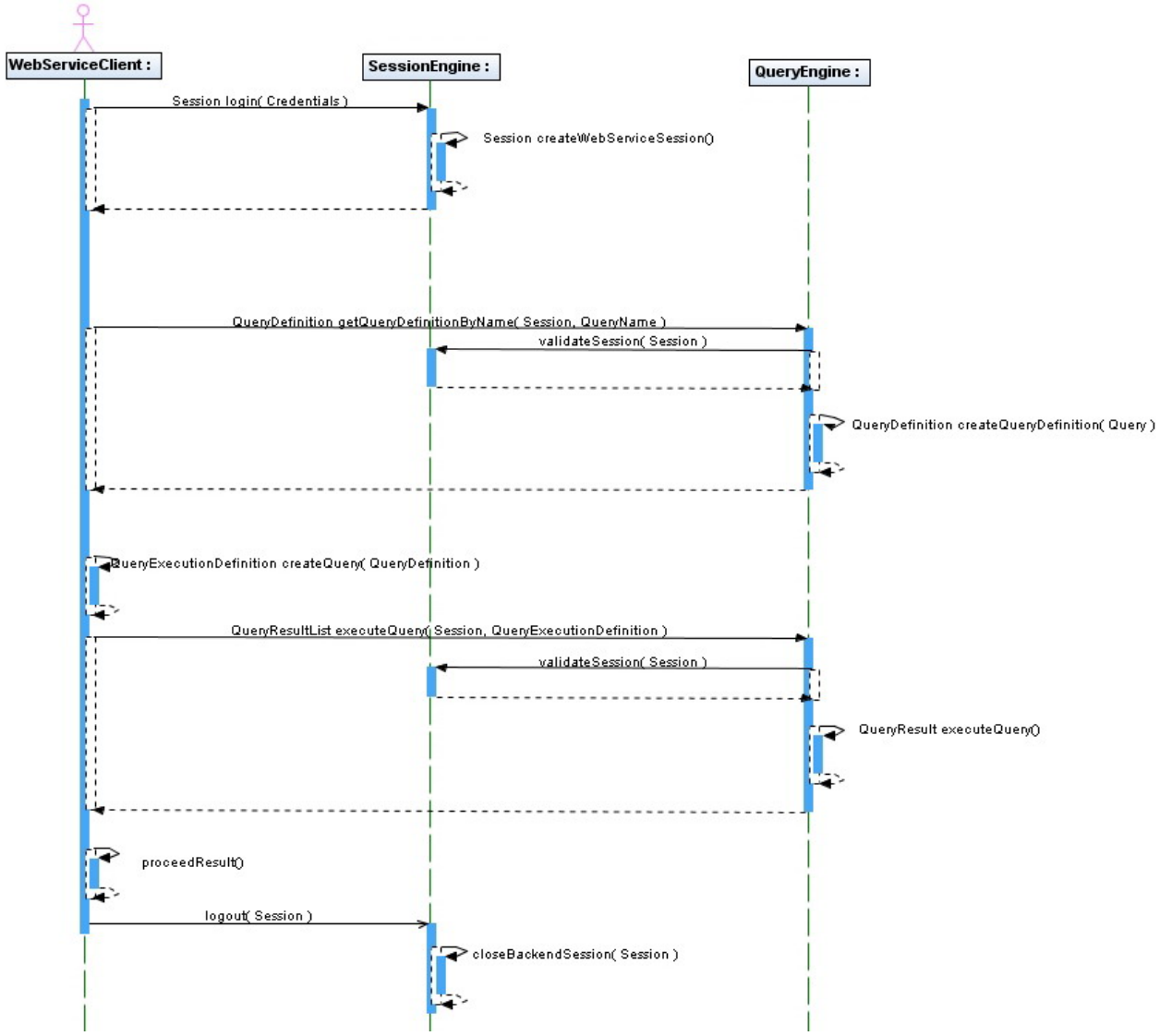

Figure 0-5 Execute Query Call Sequence

1. Obtain a web service session and authenticate with an existing IUCLID 5 user account
2. Select the query you wish to execute.

   To create a query you have these options:

   a. Load the query definition by its **name** id see 3.2.1 getAllQueryNames

      Load the query definition by its **resulting document type** see getQueryDefinitionsByResultDocumentType

   b. Create you own query execution definition if you know **exactly** what you have to do. You need to know the query name, the exact parameter names and display type see the Query Definitions for details.

3. Execute the query by using the received Session and the newly create execution definition
4. Process the result list which contains the document reference

4. Document Access Engine

All data object in the IUCLID database are uniquely identified by a pair of UUIDs which are known as DocumentReferencePK and DocumentReferencePKWs respectively. If you know this key for a single data object you can:

- read (get) the document via the web-service interface
- delete (drop) the document via the web-service interface

It is also possible to write a document with a DocumentReferencePK into the IUCLID database.

- update (overwrite) a document in the IUCLID database if the UUID already exists
- insert (create) a document in the IUCLID database if the UUID not yet already

This service is very powerful as it exposes a very generic internal API. This internal API is designed to manage all types of documents that pass through the system: substances, reference substances, templates, endpoints, dossiers, etc.

However, with great power comes great responsibility. Being a system API, no interactive safeguards like when importing an .i5z archive is available. Instead, while the API can take precautions, it still has to allow operations that can cause data loss if applied carelessly. This means also that it is very important for developers to be conversant with the official IUCLID 5 XML Format, which could be downloaded from the official IUCLID 5 website (see http://iuclid.eu/index.php?fuseaction=home.format&type=public). This format describes e.g. how to link an Annotation to its parent document (Substance, Mixture, etc.).

Note:
For more information on how to create these identifiers, please refer to section 14.1 Document Identifiers – DocumentReferencePKWs, which is part of chapter 14 Annex.

4.1. Short Introduction to Data Model

- DocumentWS
  contains the IUCLID 5 document.
  - documentReference, the document reference of the containing XML document (see Figure 0-4 Query Result List Structure).
  - Content, the XML representation of the IUCLID 5 document.

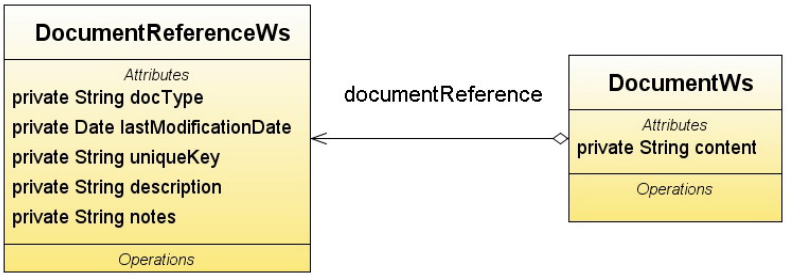

Figure 0-6 Document WS Structure

4.2. Interface Methods

This section describes each single method of the Document Access Engine. These descriptions do not explain any use case scenarios, such information can be found in section 4.3.

4.2.1. existsDocument

Purpose	Verify if the Unique Key points to an existing document.
IN	• The open (logged in) session as <tns:Session> • The Document Reference PK (<tns:documentReferencePK uniqueKey="UUID/SnapshotUUID">
OUT	• A Boolean flag <tns:isExisting>true</tns:isExisting>
Exceptions	• DocumentAccessEngineNotAvailableFault Is thrown when e.g. the back-end service is not available to fulfil the request • DocumentAccessEngineFault Is thrown if Document Access Engine is not able to fulfil your request. This may happen if the Engine caught a back-end Exception • DocumentNotFoundFault Is a tagged exception used to inform the caller that the requested document cannot be found • DocumentAccessFault Is a tagged exception used to tell the caller that the desired access to a document is prohibited due to sharing restriction

4.2.2. determineDocumentType

Purpose	Return the Type of the associated document by its UniqueKey
IN	• The open (logged in) session as <tns:Session> • The Document Reference PK (<tns:documentReferencePK uniqueKey=”UUID/SnapshotUUID”>
OUT	• The document type of the given DocumentReferencePK <tns:documentType type="LegalEntity"/>
Exceptions	• DocumentAccessEngineNotAvailableFault Is thrown when e.g. the back-end service is not available to fulfil the request • DocumentAccessEngineFault Is thrown if Document Access Engine is not able to fulfil your request. This may happen if the Engine caught a back-end Exception • DocumentNotFoundFault Is a tagged exception used to inform the caller that the requested document cannot be found

4.2.3. closeDocument

Purpose	Release the reader or writer lock of the current session on document the specified by the given UniqueKey. **Note:** Uncommitted changes will be lost from the server-side point of view.
IN	• The open (logged in) session as <tns:Session> • The Document Reference PK (<tns:documentReferencePK uniqueKey=”UUID/SnapshotUUID”>
OUT	• Boolean (direct Boolean xml binding) <tns:success>true</tns:success> o True to explicitly notify that everything was OK otherwise will throw a detailed exception
Exceptions	• DocumentAccessEngineNotAvailableFault Is thrown when e.g. the back-end service is not available to fulfil the request • DocumentNotFoundFault Is a tagged exception used to inform the caller that the requested document cannot be found • DocumentAccessFault Is a tagged exception used to tell the caller that the desired access to a document is prohibited due to sharing restriction

4.2.4. fetchDocument

Purpose	Fetch a document without setting any access locks. This means that no lock would be created on the server side. Obviously this might return the same result as 4.2.5 openDocumentForReading but fetchDocument will close the document immediately and the web service customer does not have to do this. If you are using openDocumentForReading you have to call closeDocument after you have finished your process. The structure of the XML document fits to the official public export format of the IUCLID 5 System.
IN	• The open (logged in) session as <tns:Session> • The Document Reference PK (<tns:documentReferencePK uniqueKey="UUID/SnapshotUUID">
OUT	• The Web service document (<tns:document>) contains o The Document Reference o The IUCLID 5 document content <tns:document> <tns:documentReference type="LegalEntity" ... /> <tns:content> <i5:LegalEntity ... </tns:content> </tns:document>
Exceptions	• DocumentAccessEngineNotAvailableFault Is thrown when e.g. the back-end service is not available to fulfil the request • DocumentAccessEngineFault Is thrown if Document Access Engine is not able to fulfil your request. This may happen if the Engine caught a back-end Exception • DocumentNotFoundFault Is a tagged exception used to inform the caller that the requested document cannot be found • DocumentAccessFault Is a tagged exception used to tell the caller that the desired access to a document is prohibited due to sharing restriction

4.2.5. openDocumentForReading

Purpose	Open a document for read-only access and acquiring the read lock at the server. The read lock would be present until you call closeDocument (see 4.2.3).
IN	• The open (logged in) session as <tns:Session> • The Document Reference PK (<tns:documentReferencePK uniqueKey=”UUID/SnapshotUUID”>
OUT	• The Web service document (<tns:document>) contains o The Document Reference o The IUCLID 5 document content <tns:document> <tns:documentReference type="LegalEntity" ... /> <tns:content> <i5:LegalEntity ... </tns:content> </tns:document>
Exceptions	• DocumentAccessEngineNotAvailableFault Is thrown when e.g. the service is not available to fulfil the request • DocumentAccessEngineFault Is thrown if Document Access Engine is not able to fulfil your request. This may happen if the Engine caught a back-end Exception. • DocumentNotFoundFault Is a tagged exception used to inform the caller that the requested document cannot be found. • DocumentAccessFault Is a tagged exception used to tell the caller that the desired access to a document is prohibited due to sharing restriction

4.2.6. compareDocuments

Purpose	Compare the specified remote Document to the local server content. As result you will get a simple notification which declares the state. • **INDETERMINATE**: The comparison status has not (yet) been determined • **NOTFOUND**: The remote document does not have a local counterpart with the same identifier • **IDENTICAL**: The documents are absolutely identical • **OLDER**: The documents are unequal, the remote document is older than the local document • **NEWER**: The documents are unequal, the remote document is newer than the local document • **CONFLICT**: The documents are unequal, although the last-modification are identical (filtered documents) • **LOCKED**: The documents could not be compared because the local document was currently locked • **ACCESSDENIED**: The documents could not be compared because the local document is access protected • **VALIDATION_VIOLATION**: The documents could not be compared because validation detected a violation • **ERROR**: The documents could not be compared because of an unexpected error
IN	• The open (logged in) session as <tns:Session> • The Web service document (<tns:document>) o The IUCLID 5 document content <tns:document> <tns:content> <i5:LegalEntity ... </tns:content> </tns:document>
OUT	• Document Comparison State <tns:documentComparisonStatus comparisonStatus="IDENTICAL" />
Exceptions	• DocumentAccessEngineNotAvailableFault Is thrown when e.g. the back-end service is not available to fulfil the request • DocumentAccessEngineFault Is thrown if Document Access Engine is not able to fulfil your request. This may happen if the Engine caught a back-end Exception. • DocumentAccessFault Is a tagged exception used to tell the caller that the desired access to a document is prohibited due to sharing restriction

4.2.7. openDocumentForWriting

Purpose	Open a document for read-write access and acquiring the write lock at the server. The write lock would be present until you call closeDocument (see 4.2.3).
IN	• The open (logged in) session as <tns:Session> • The Document Reference PK (<tns:documentReferencePK uniqueKey="UUID/SnapshotUUID">
OUT	• The Web service document (<tns:document>) contains o The Document Reference o The IUCLID 5 document content <tns:document> <tns:documentReference type="LegalEntity" ... /> <tns:content> <i5:LegalEntity ... </tns:content> </tns:document>
Exceptions	• DocumentAccessEngineNotAvailableFault Is thrown when e.g. the back-end service is not available to fulfil the request • DocumentAccessEngineFault Is thrown if Document Access Engine is not able to fulfil your request. This may happen if the Engine caught a back-end Exception. • DocumentNotFoundFault Is a tagged exception used to inform the caller that the requested document cannot be found. • DocumentAccessFault Is a tagged exception used to tell the caller that the desired access to a document is prohibited due to sharing restriction

4.2.8. commitDocument

Purpose	Commit the specified document to the IUCLID 5 Server. The requestor must have the exclusive **write lock** on the Document which could be obtained via open the document for writing (see *4.2.7 openDocumentForWriting*). **Note:** the lock state will not be modified by this call. The document remains open after the operation! If you like to release the write lock, you have to call *closeDocument*.
IN	• The open (logged in) session as <tns:Session> • The Web service document (<tns:document>) o The IUCLID 5 document content <tns:document> <tns:content> <i5:LegalEntity ... </tns:content> </tns:document>
OUT	Boolean (direct Boolean xml binding) True to explicitly notify that everything was OK, otherwise will throw a detailed exception <tns:success>true</tns:success>
Exceptions	• DocumentAccessEngineNotAvailableFault Is thrown when e.g. the back-end service is not available to fulfil the request • DocumentAccessEngineFault Is thrown if Document Access Engine is not able to fulfil your request. This may happen if the Engine caught a back-end Exception • DocumentNotFoundFault Is a tagged exception used to inform the caller that the requested document cannot be found • DocumentAccessFault Is a tagged exception used to tell the caller that the desired access to a document is prohibited due to sharing restriction

4.2.9. deleteDocument

Purpose	Delete the document referenced by the specified document reference. **Note:** the server call can only be successful if no lock remains on the requested document. Therefore, no such lock can be present in the local handle table if the server call succeeds and this manager works correctly.
IN	• The open (logged in) session as <tns:Session> • The Document Reference PK (<tns:documentReferencePK uniqueKey=”UUID/SnapshotUUID”>
OUT	• Boolean (direct Boolean xml binding) <tns:success>true</tns:success> o True to explicitly notify that everything was OK otherwise will throw a detailed exception
Exceptions	• DocumentAccessEngineNotAvailableFault Is thrown when e.g. the back-end service is not available to fulfil the request • DocumentNotFoundFault Is a tagged exception used to inform the caller that the requested document cannot be found • DocumentAccessFault Is a tagged exception used to tell the caller that the desired access to a document is prohibited due to sharing restriction

4.2.10. createDocument

Purpose	Creates a new Document object for the given document type. **Note:** The document has open write access to the document after successful call to this method, so to finish working with the document a call to *4.2.3 closeDocument* is required. The created document would only be saved at the database layer if you call 4.2.8 commitDocument.
IN	• The open (logged in) session as <tns:Session> • Document Tye (Substance, Mixture, Dossier) <tns:documentType type="LegalEntity"/>
OUT	• The Web service document (<tns:document>) contains o The Document Reference o The IUCLID 5 document content <tns:document> <tns:documentReference type="LegalEntity" ... /> <tns:content> <i5:LegalEntity ... </tns:content> </tns:document>
Exceptions	• DocumentAccessEngineNotAvailableFault Is thrown when e.g. the back-end service is not available to fulfil the request • DocumentCreationFault is thrown to tell the caller that the desired creation process of a document had an error. • DocumentAccessFault Is a tagged exception used to tell the caller that the desired access to a document is prohibited due to sharing restriction

4.3. How to use the Document Access Engine

The following UML Sequence Diagram demonstrates one possible usage case scenario.

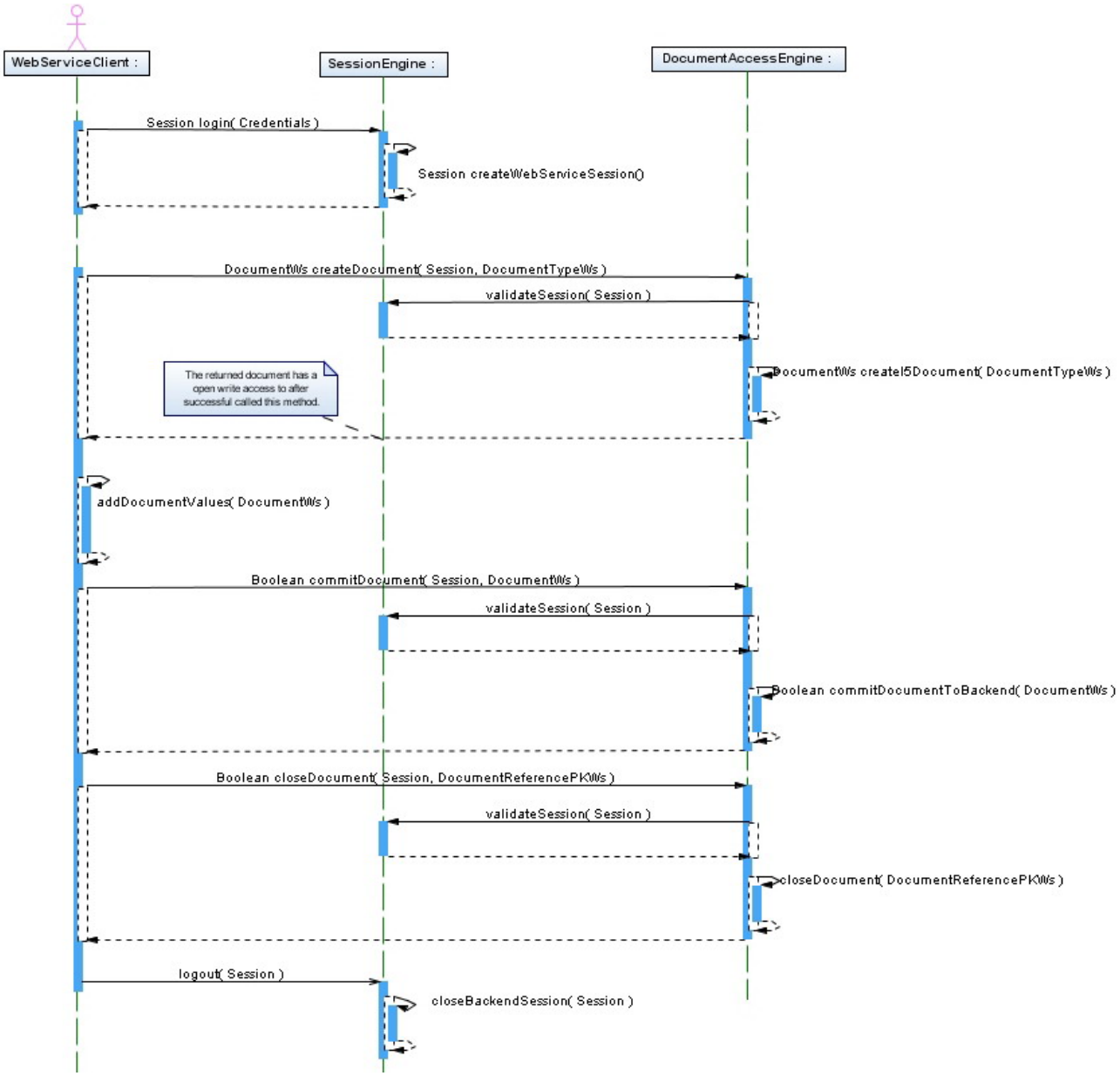

Figure 0-7 Create and Commit New Document

1. Obtain a web service session and authenticate with an existing IUCLID 5 user account
2. Call the Document Access Engine to create the Document. As result you will get an empty document structure for the requested document type.
3. This document has already a read-write lock, so you can exclusively change the content.
4. Commit the document so your changes will be come persistent.

   Note: after committing a document no lock will be released!

5. Close the document to release all held locks.
6. Call logout to close the back-end connection.

5. Endpoint Access Engine

The Endpoint Access Engine offers convenient operations for endpoint (endpoint record, endpoint study record and endpoint summary) handling. Normally, endpoint operations modify the endpoint document and the enclosing container (template, substance, mixture). The operations of the Endpoint Access Engine take care of all necessary steps when handling endpoints.

5.1. Interface Methods

This section describes the operations of the Endpoint Access Engine.

5.1.1. createNewEndpoint

Purpose	Create a new endpoint of the given type and add the created endpoint to an endpoint container document (substance, mixture or template)
IN	• The open (logged in) session as <tns :session> • Endpoint container document (substance, mixture or template) as <tns:document> • The valid endpoint kind (e.g. EC_ABOVE_GROUND_TOX) as string according to the IUCLID 5 format. • The name of the endpoint document as string.
OUT	• The updated endpoint container document (substance, mixture or template) as <tns:document>. The caller must make sure to use this in subsequent calls. To finalize the creation of the endpoint, this document has to be committed. • The reference of the created endpoint document as <tns:documentReference>
Prerequisites	The endpoint container document (substance, mixture or template) must be open for writing for the session that is used to create a new endpoint.
Exceptions	• EndpointAccessEngineNotAvailableFault Is thrown when e.g. the back-end service is not available to fulfil the request • EndpointAccessEngineFault Is thrown if Endpoint Access Engine is not able to fulfil your request. This may happen if the Engine caught a back-end Exception

5.1.2. createNewEndpointSummary

Purpose	Create a new endpoint summary of the given type and add the created endpoint summary to an endpoint container document (substance, mixture or template)
IN	• The open (logged in) session as <tns :session> • Endpoint container document (substance, mixture or template) as <tns:document> • The valid endpoint summary kind (e.g. EC_ABOVE_GROUND_TOX) as string according to the IUCLID 5 format. • The name of the endpoint summary document as string.
OUT	• The updated endpoint container document (substance, mixture or template) as <tns:document>. The caller must make sure to use this in subsequent calls. To finalise the creation of the endpoint summary, this document has to be committed. • The reference of the created endpoint summary document as <tns:documentReference>.
Prerequisites	The endpoint container document (substance, mixture or template) must be open for writing for the session that is used to create a new endpoint.
Exceptions	• EndpointAccessEngineNotAvailableFault Is thrown when e.g. the back-end service is not available to fulfil the request • EndpointAccessEngineFault Is thrown if Endpoint Access Engine is not able to fulfil your request. This may happen if the Engine caught a back-end Exception

5.1.3. copyEndpoint

Purpose	Copies an endpoint from one endpoint container document (substance, mixture or template) to another endpoint container document.
IN	• The open (logged in) session as <tns :session> • The target endpoint container document (substance, mixture or template) as <tns:document> • The unique key of the source endpoint as <tns:documentReferencePK >
OUT	• The updated endpoint container document (substance, mixture or template) as <tns:document>. The caller must make sure to use this in subsequent calls. To finalize the copying of the endpoint, this document has to be committed. • The reference of the copied endpoint document as <tns:documentReference>.
Prerequisites	The target endpoint container document (substance, mixture or template) must be open for writing for the session that is used to copy an endpoint.
Exceptions	• EndpointAccessEngineNotAvailableFault Is thrown when e.g. the back-end service is not available to fulfil the request • EndpointAccessEngineFault Is thrown if Endpoint Access Engine is not able to fulfil your request. This may happen if the Engine caught a back-end Exception

5.1.4. copyEndpoints

Purpose	Copies multiple endpoints from one endpoint container document (substance, mixture or template) to another endpoint container document.
IN	• The open (logged in) session as <tns :session> • The target endpoint container document (substance, mixture or template) as <tns:document> • The unique keys of the source endpoints as <tns:documentReferencePK >
OUT	• The updated endpoint container document (substance, mixture or template) as <tns:document>. The caller must make sure to use this in subsequent calls. To finalize the copying of the endpoints, this document has to be committed. • The references of the copied endpoint documents as <tns:documentReference>.
Prerequisites	The target endpoint container document (substance, mixture or template) must be open for writing for the session that is used to copy endpoints.
Exceptions	• EndpointAccessEngineNotAvailableFault Is thrown when e.g. the back-end service is not available to fulfil the request • EndpointAccessEngineFault Is thrown if Endpoint Access Engine is not able to fulfil your request. This may happen if the Engine caught a back-end Exception

5.1.5. deleteEndpoint

Purpose	Delete a single endpoint from persistence. The reference to the endpoint is also automatically removed from its parent document. (substance, mixture or template)
IN	• The open (logged in) session as <tns :session> • The endpoint container document (substance, mixture or template) as <tns:document> • The unique key of the endpoint to be deleted as <tns:documentReferencePK >
OUT	• The updated endpoint container document (substance, mixture or template) as <tns:document>. The caller must make sure to use this in subsequent calls.
Prerequisites	• The endpoint container document (substance, mixture or template) must be open for writing for the session.
Exceptions	• EndpointAccessEngineNotAvailableFault Is thrown when e.g. the back-end service is not available to fulfil the request • EndpointAccessEngineFault Is thrown if Endpoint Access Engine is not able to fulfil your request. This may happen if the Engine caught a back-end Exception

5.1.6. deleteEndpoints

Purpose	Delete a multiple endpoint from persistence. The reference to the endpoint is also automatically removed from its parent document (substance, mixture or template).
IN	• The open (logged in) session as <tns :session> • The endpoint container document (substance, mixture or template) as <tns:document> • The unique keys of the endpoints to be deleted as <tns:documentReferencePK >
OUT	• The updated endpoint container document (substance, mixture or template) as <tns:document>. The caller must make sure to use this in subsequent calls.
Prerequisites	• The endpoint container document (substance, mixture or template) must be open for writing for the session.
Exceptions	• EndpointAccessEngineNotAvailableFault Is thrown when e.g. the back-end service is not available to fulfil the request • EndpointAccessEngineFault Is thrown if Endpoint Access Engine is not able to fulfil your request. This may happen if the Engine caught a back-end Exception

5.1.7. referenceEndpoint

Purpose	Reference the specified endpoint to the specified new owner (substance, mixture or template) document by copying it and establishing a special local reference to the original (filling in the .origin attribute).
IN	• The open (logged in) session as <tns :session> • The endpoint container document (substance, mixture or template) as <tns:document> • The unique key of the endpoint to be referenced (the source of the reference) as <tns:documentReferencePK > • A note to be added to the origin reference
OUT	• The updated endpoint container document (substance, mixture or template) as <tns:document>. The caller must make sure to use this in subsequent calls. • The unique keys of the new endpoint reference as <tns:documentReferencePK >
Prerequisites	• The endpoint container document (substance, mixture or template) must be open for writing for the session. • The specified endpoint document to be referenced and all associated attachments must be readable.
Exceptions	• EndpointAccessEngineNotAvailableFault Is thrown when e.g. the back-end service is not available to fulfil the request • EndpointAccessEngineFault Is thrown if Endpoint Access Engine is not able to fulfil your request. This may happen if the Engine caught a back-end Exception

5.1.8. referenceEndpoints

Purpose	Reference the specified endpoints to the specified new owner (substance, mixture or template) document by copying it and establishing a special local reference to the original (filling in the .origin attribute).
IN	• The open (logged in) session as <tns :session> • The target endpoint container document (substance, mixture or template) as <tns:document> • The unique keys of the endpoinst to be referenced (the source of the reference) as <tns:documentReferencePK > • A note to be added to the origin references
OUT	• The updated endpoint container document (substance, mixture or template) as <tns:document>. The caller must make sure to use this in subsequent calls. • The unique keys of the new endpoint references as <tns:documentReferencePK >
Prerequisites	• The endpoint container document (substance, mixture or template) must be open for writing for the session. • The specified endpoint documents to be referenced and all associated attachments can be read.
Exceptions	• EndpointAccessEngineNotAvailableFault Is thrown when e.g. the back-end service is not available to fulfil the request • EndpointAccessEngineFault Is thrown if Endpoint Access Engine is not able to fulfil your request. This may happen if the Engine caught a back-end Exception.

5.1.9. updateEndpoint

Purpose	Update the specified referencing endpoint document accordingly to the content of its original endpoint.
IN	• The open (logged in) session as <tns :session> • The endpoint container document (substance, mixture or template) as <tns:document> • The endpoint reference document to be updated as <tns:document>
OUT	• The modified endpoint container document (substance, mixture or template) as <tns:document> • The modified endpoint reference document as <tns:document>
Prerequisites	• The endpoint container document (substance, mixture or template) must be open for writing for the session. • The specified endpoint documents to be updated must be open for writing for the session.
Exceptions	• EndpointAccessEngineNotAvailableFault Is thrown when e.g. the back-end service is not available to fulfil the request • EndpointAccessEngineFault Is thrown if Endpoint Access Engine is not able to fulfil your request. This may happen if the Engine caught a back-end Exception.

5.1.10. detachEndpoint

Purpose	Detach the referencing endpoint from its original endpoint document. The reference to the endpoint is also automatically updated and the endpoint container document (substance, mixture or template) is modified. After the operation, two separated, independent endpoints exist.
IN	• The open (logged in) session as <tns :session> • The endpoint container document (substance, mixture or template) as <tns:document> • The endpoint reference document to be detached as <tns:document>
OUT	• The modified endpoint container document (substance, mixture or template) as <tns:document>. • The modified endpoint reference document as <tns:document>
Prerequisites	• The endpoint container document (substance, mixture or template) must be open for writing for the session. • The specified endpoint documents to be updated must be open for writing for the session.
Exceptions	• EndpointAccessEngineNotAvailableFault Is thrown when e.g. the back-end service is not available to fulfil the request • EndpointAccessEngineFault Is thrown if Endpoint Access Engine is not able to fulfil your request. This may happen if the Engine caught a back-end Exception.

5.2. How to use the Endpoint Access Engine

The purpose of the endpoint access engine is to provide convenient operations for endpoint handling.

The operations of the endpoint access engine typically modify the endpoint container document (substance, mixture or template) and one or more endpoint documents or the references of the endpoint documents. It is important to use the references and documents returned in the results of these operations to ensure that the IUCLID 5 database stays in a consistent state. Always make sure to update your endpoint container documents and endpoint documents from the results.

The endpoint access engine commits changes to endpoint container and endpoint documents to the database.

6. Referential Integrity Engine

The referential integrity engine is typically needed in an import scenario. The referential integrity engine is then used as a last step after all documents have been created in the IUCLID 5 database and ensures that a set of documents is linked correctly. Additionally it updates the dossier search index (for finding dossiers through queries), if dossiers are created / handled.

The referential integrity engine executes the following checks:

- Checks for endpoints (endpoint records, endpoint study records, endpoint summaries):
  - Check if “ownerRef” is resolvable. If not: delete the endpoint.
  - Check if “ownerRef” points references a valid type (substance, mixture or template). If not: delete the endpoint.
  - Check if the endpoint container that is referenced by the “ownerRef” references the endpoint

- Checks endpoint containers (substance, micture or templates), if they correctly reference endpoints that reference the endpoint container through their “ownerRef”. If not: correct the “ownerRef”.
- Update the dossier search index.
- Check if attachments are correctly linked. Delete orphaned attachments.
- Check if annotations are correctly linked. Delete orphaned attachments.

If the check fails, the referential integrity engine corrects the detected problem automatically.

6.1. Interface Methods

This section describes the operations of the Referential Integrity Engine.

6.1.1. check

Purpose	Check the referential integrity of the given documents.
IN	• The open (logged in) session as <tns :session> • A list of items to include in the referential integrity check. The items contain The unique keys of the documents to check as <tns:documentReferencePK > and the type of the documents to check as <tns:documentType>
OUT	n.a.
Prerequisites	No document contained in the list of items to check must be locked by the web service during the operation. All documents to be checked have to be closed. All locking and modification operations are completely handled by the server.
Exceptions	• ReferentialIntegrityEngineNotAvailableFault Is thrown when e.g. the back-end service is not available to fulfil the request • ReferentialIntegrityEngineFault Is thrown if Referential Integrity Engine is not able to fulfil your request. This may happen if the Engine caught a back-end Exception.

6.2. How to use the Referential Integrity Engine

The purpose of the referential integrity engine is to provide operations to be used in a typical import scenario, where it might be necessary to do some corrective actions after the imported documents have been created in the database. It ensures that orphaned links between documents are handled correctly and that the linkage between documents (especially endpoints and their containers) is valid.

The referential integrity engine is used for maintaining referential integrity across document boundaries, e.g. if a company is deleted, also its associated sites have to be deleted also.

The referential integrity engine must always be called after the web service has committed the contents of an IUCLID 5 i5z file as documents to the IUCLID 5 database (i.e. if the web service client is implementing an import operation).

7. Query Tool Engine

Please note that the web service interface for the Query Tool Engine will only work when the IUCLID 5 Query Tool server-side plug-in is installed on the IUCLID 5 distributed version server. Otherwise, the interface methods of the Query Tool Engine will abort with an exception.

7.1. Interface Methods

This section describes the operations of the Query Tool Engine.

7.1.1. countQuery

Purpose	Count the number of result items that are produced when the specified query expression is executed. Note that counting does not support the block combination logic, so only simple query blocks can be contained in the query expression to be counted.
IN	• The open (logged in) session as <tns :session> • The query expression to count as <tns: QueryExpression>. Note that a query expression specified can only contain simple query blocks. Logical operations that are combining query blocks are not supported by countQuery.
OUT	The expected number of result items that will be found when the query expression specified will be executed.
Prerequisites	The specified query expression can only contain simple query blocks and no logical operations.
Exceptions	• QueryToolEngineNotAvailableFault Is thrown when e.g. the back-end service is not available to fulfil the request • QueryToolEngineFault Is thrown if the Query Tool Engine is not able to fulfil your request. This may happen if the Engine caught a back-end Exception.

7.1.2. executeQuery

Purpose	Execute the given query expression, which may contain simple blocks as well as block conjunction logic, and return the found query result.
IN	• The open (logged in) session as <tns :session> • The query expression to execute as <tns: QueryExpression>.
OUT	The query result which contains a list of query result items (containing the <tns:documentReferencePK> of indexed items, parent documents, dossiers, etc.) as well as original values as stored in the query index and the query fields in the query expression that produced the result item.
Prerequisites	none
Exceptions	• QueryToolEngineNotAvailableFault Is thrown when e.g. the back-end service is not available to fulfil the request • QueryToolEngineFault Is thrown if the Query Tool Engine is not able to fulfil your request. This may happen if the Engine caught a back-end Exception.

7.2. Data Structures

7.2.1. Query Expression

The following figure shows the structure of the query expression. A QueryExpression is built with QueryBlock items that can be SimpleEndpointQueryBlock items or SimpleQueryBlock items. The SimpleEndpointQueryBlock items find endpoint and endpoint summary related data. The SimpleQueryBlock items find non-endpoint related data. The query blocks can be combined with LogicalExpressionQueryBlock items that support the block conjunction logic as applied by the IUCLID 5 query tool, offering AND, OR and NOT operations between blocks.

SimpleEndpointQueryBlock items specify the document type of endpoint searched (Endpoint record, endpoint study record or endpoint summary record), the endpoint kind (e.g. PC_BOILING) and a field restriction criteria that can be used to restrict the query result items in case no query field is specified (and thus no field names are available otherwise).

SimpleQueryBlock items specify the document type of the document searched and a field restriction criterion that can be used to restrict the query result items in case no query field is specified (and thus no field names are available otherwise).

The simple query blocks consist of QueryField items (possible types are strings – which also represent the IUCLID 5 pick list values in clear text- , dates, numeric values and ranges). The field item specifies which index field to search and an ID, which can be used for identification purposes. Note that the fieldname, which specifies the index field to search, can contain database wildcards. This makes it possible to build queries like “search for all annotations to endpoint documents (without specifying the section/endpoint kind)”.

Each of the query fields specifies the possible query values. If a query field specifies more than one query value, the values are automatically combined with an “OR” operation when searching the query tool index.

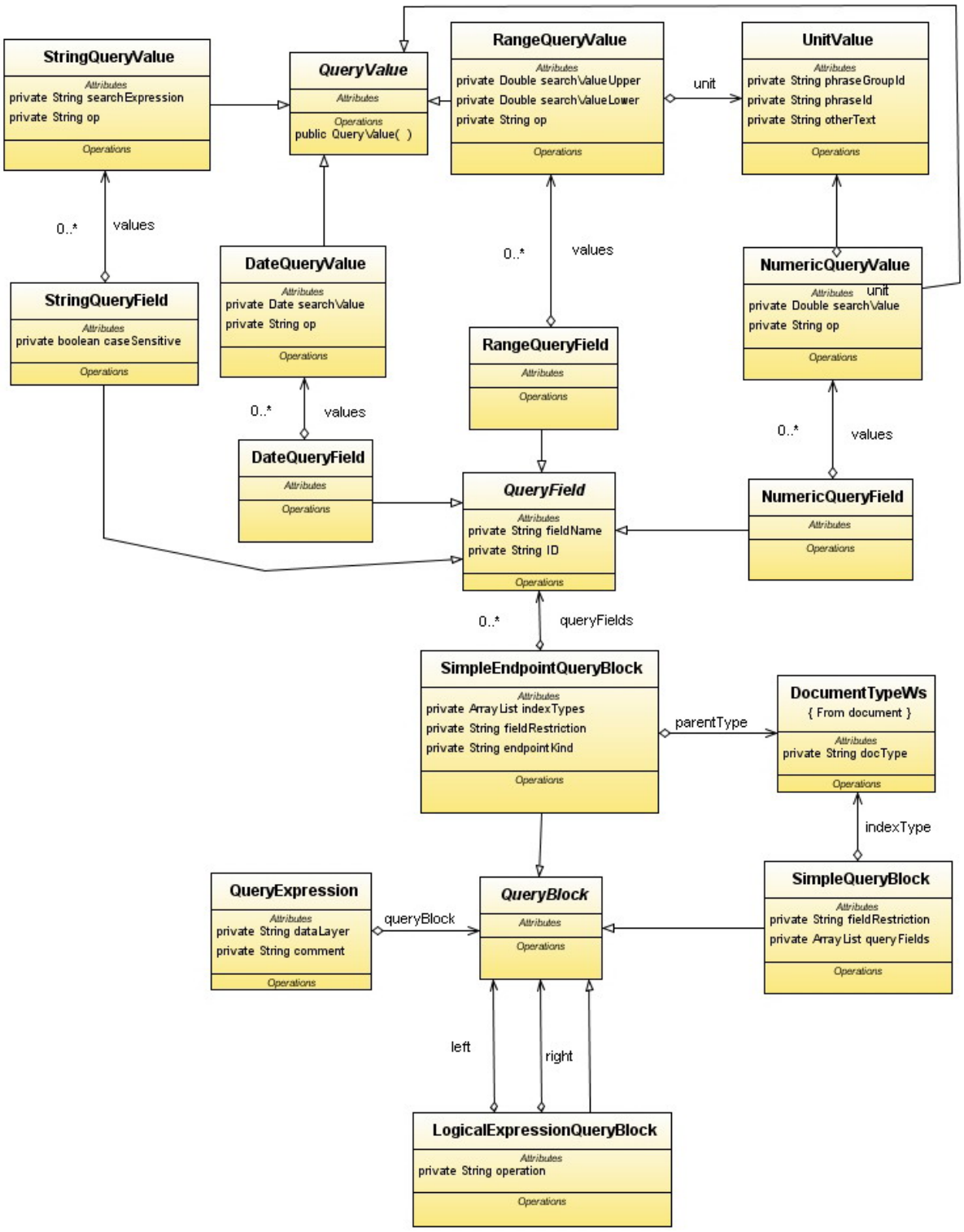

Figure 0-8 Structure of Query Expression

7.2.2. Query Result

The following figure shows the structure of the query result. The query result contains query result items that can be used to access the documents that contain the found data. The query result items reference the values that were found in the index when executing the query. These values are referencing the fields contained in the executed query expression that found this value in the query index.

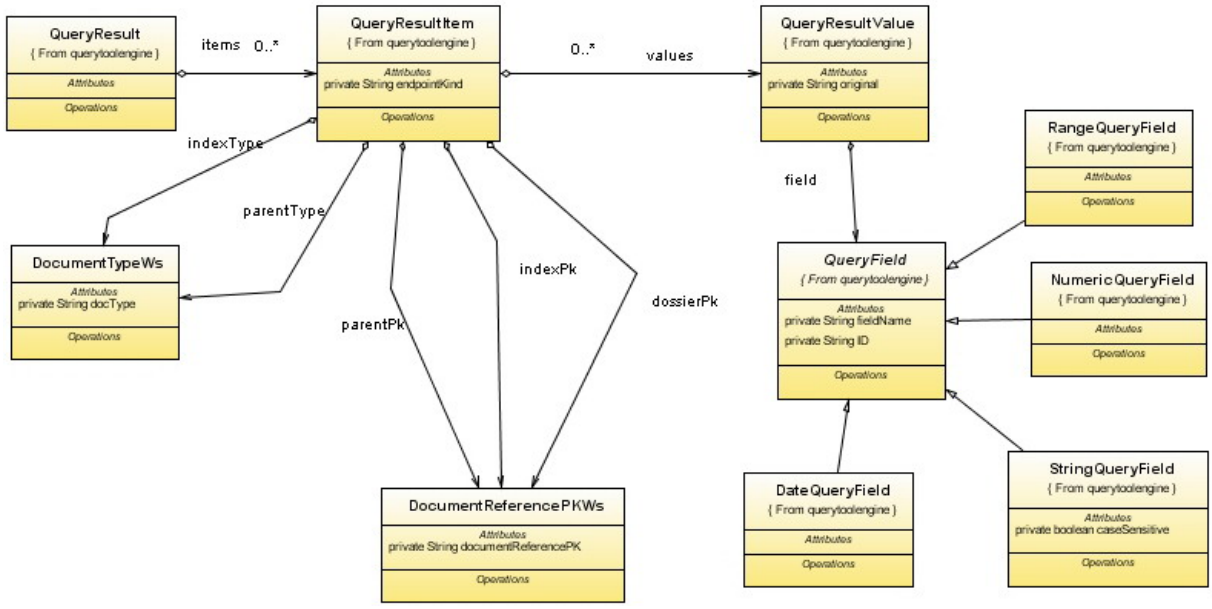

Figure 0-9 Structure of Query Result

7.3. How to use the Query Tool Engine

The recommended approach for building query expression is to use the IUCLID 5 query tool on the IUCLID 5 client to create the query expression. Then export the created query expression and translate the query expression to program for the Query Tool Engine web service. This approach is recommended, because the exact naming for indexed IUCLID 5 data fields and the rules applied when indexing IUCLID 5 data, as well as data type information about the index entries, are information only available to the query tool itself. Exporting the queries and using the exported queries as input for implementation ensures that the correct naming is applied, as the exported file contains the fieldnames and phrase information for units as well as string constants necessary for logical expressions and operators.

This section shows how to translate a query from an export file to program code accessing the web service.

Example: a query that finds all substances in the raw data layer which contain a melting point of 100°C and a boiling point of 100°C.

The exported query (in XML format contained in a file with extension “i5q”) looks like:

```
<?xml version="1.0"?>
<!-- Created: 2008-10-20 12:17:01 by QueryUIPlugin; 5.0.0 -->
<expression>
  <version>5.0.0</version>
  <comment/>
  <datalayer>RAW_DATA</datalayer>
  <query level="1">
    <path>
      <pathelement id="TR.SE00.00"/>
      <pathelement id="TR.SU.SE00.00"/>
      <pathelement id="Q.S.04.00.00"/>
      <pathelement id="SE04.02"/>
    </path>
    <criteria>
      <range id="SE04.02.00.0470" fieldname="PC_MELTING.MELTINGPT.PRECISION_LOQUALIFIER">
        <value operator="ONLY_EXACT_MATCH" lower="100.0">
          <unit phrasegroup="A102" phrase="2493"/>
        </value>
      </range>
    </criteria>
  </query>
  <AND level="1"/>
  <query level="1">
    <path>
      <pathelement id="TR.SE00.00"/>
      <pathelement id="TR.SU.SE00.00"/>
      <pathelement id="Q.S.04.00.00"/>
      <pathelement id="SE04.03"/>
    </path>
    <criteria>
      <range id="SE04.03.00.0470" fieldname="PC_BOILING.BOILINGPT.PRECISION_LOQUALIFIER">
        <value operator="ONLY_EXACT_MATCH" lower="100.0">
          <unit phrasegroup="A102" phrase="2493"/>
        </value>
      </range>
    </criteria>
  </query>
</expression>
```

XML file 0-2 Query Export

The information highlighted in the file contents above are necessary for building the query expression program code for the web service.
The following code highlights the places where this information is needed in the same colour.

```
// build a melting point and a boiling point endpoint query block
final SimpleEndpointQueryBlock meltingQueryBlock = new SimpleEndpointQueryBlock();
final SimpleEndpointQueryBlock boilingQueryBlock = new SimpleEndpointQueryBlock();

// build executable query expression

final QueryExpression expression = new QueryExpression();
expression.setComment("test expression");
expression.setDataLayer("RAW_DATA");

 // combine query blocks
final LogicalExpressionQueryBlock logicBlock  = new LogicalExpressionQueryBlock();
logicBlock.setOperation(“AND”);
logicBlock.setLeft(meltingQueryBlock);
logicBlock.setRight(boilingQueryBlock);

expression.setQueryBlock(logicBlock);
```

```
// fill query blocks

// index types and parent types
final ArrayList<DocumentTypeWs> indexTypes = new ArrayList<DocumentTypeWs>();


indexTypes.add(new DocumentTypeWs( "EndpointStudyRecord" ));

final DocumentTypeWs parentType = new DocumentTypeWs( "Substance" );
// melting point

meltingQueryBlock.setEndpointKind("PC_MELTING");
meltingQueryBlock.setIndexTypes(indexTypes);
meltingQueryBlock.setParentType(parentType);

final ArrayList<QueryField> meltingQueryFields = new ArrayList<QueryField>();
meltingQueryBlock.setQueryFields(meltingQueryFields);

final RangeQueryField meltingQF = new RangeQueryField();
meltingQF.setID("melting");
meltingQF.setFieldName("PC_MELTING.MELTINGPT.PRECISION_LOQUALIFIER");


final ArrayList<RangeQueryValue> meltingValues = new ArrayList<RangeQueryValue>();
final RangeQueryValue meltingQV = new RangeQueryValue();
meltingQV.setOp("ONLY_EXACT_MATCH");
meltingQV.setSearchValueLower(100.0);
final UnitValue meltingUnit = new UnitValue();
meltingUnit.setPhraseGroupId("A102");
meltingUnit.setPhraseId("2493");
meltingQV.setUnit(meltingUnit);
meltingValues.add(meltingQV);
meltingQF.setValues(meltingValues);

meltingQueryFields.add(meltingQF);


// boiling point

boilingQueryBlock.setEndpointKind("PC_BOILING");
boilingQueryBlock.setIndexTypes(indexTypes);
boilingQueryBlock.setParentType(parentType);

final ArrayList<QueryField> boilingQueryFields = new ArrayList<QueryField>();
boilingQueryBlock.setQueryFields(boilingQueryFields);

final RangeQueryField boilingQF = new RangeQueryField();
boilingQF.setID("boiling");
boilingQF.setFieldName("PC_BOILING.BOILINGPT.PRECISION_LOQUALIFIER");


final ArrayList<RangeQueryValue> boilingValues = new ArrayList<RangeQueryValue>();
final RangeQueryValue boilingQV = new RangeQueryValue();
boilingQV.setOp("ONLY_EXACT_MATCH");
boilingQV.setSearchValueLower(100.0);
final UnitValue boilingUnit = new UnitValue();
boilingUnit.setPhraseGroupId("A102");
boilingUnit.setPhraseId("2493");
boilingQV.setUnit(boilingUnit);
boilingValues.add(boilingQV);

boilingQF.setValues(boilingValues);

boilingQueryFields.add(boilingQF);
```

Code 0-3 Query as Pseudo Code

The type of the simple query block (SimpleEndpointQueryBlock or SimpleQueryBlock) depends on the information that is searched and where this information is stored. SimpleEndpointQueryBlock search in endpoint records, endpoint study records and endpoint summaries. SimpleQueryBlock search in substances, mixtures, templates. The simple query blocks also specify which type of index items they are scanning (by specifying the document type). This allows e.g. to build SimpleEndpointQueryBlock queries that only search for endpoint summaries.

Please note that string query values might specify special query expression values in the exported XML, that have first to be translated in multiple query values. Additionally, wildcard translation has to be done.

```
....

 <string id="S01.01.0100-R00.00.0120.3" fieldname="/referenceSubstanceRef/ecSubstanceInventoryEntryRef/casNumber">
      <value operator="LIKE" expression="2*|3*"/>
  </string>
....
```

XML file 0-4 Special String Query Expressions

The expression “2*|3*” translates into a StringQueryField that has two StringQueryValue items. The first StringQueryValue is specifying “2%” (wildcard translated to database wildcard) as search expression whereas the second StringQueryValue is specifying “3%” as search expression.

8. Inventory Engine

The inventory engine enables you to read, write, update and create Literature Inventory Entries.

Note:

As the inventory entry is not part of the IUCLID 5 document structure, you do not have a DocumentReferencePK (UUID/UUID). Each entry has a **system depending** primary key (unique for one IUCLID 5 Server)!

Two entries of **different** IUCLID 5 systems might have the same systemDependingPK, but are not the **same** entry.

You have to be very **careful** when exchanging Inventory Entries between IUCLID 5 databases!

8.1. Interface Methods

This section describes the operations of the Inventory Engine.

8.1.1. getLiteratureInventoryQuery

Purpose	Return the QueryDefinition (see 3.1 Short introduction to data model) for the Literature Inventory.
IN	• The open (logged in) session as <tns:Session>
OUT	• The Query Definition (<tns:queryDefinition>) of the Literature Inventory (see Figure 0-2 Query Definition Structure)
Exceptions	• InventoryEngineNotAvailableFault Is thrown when e.g. the service is not available to fulfil the request • InventoryEngineFault Is thrown if Inventory Engine is not able to fulfil your request. This may happen if the Engine caught a back-end Exception.

The following table shows the query definition (see also Table 0-2 Mapping Query Parameter Display Type – Query Parameter Value Type).

- Query Name = find_complex_literatureinventoryentries
- Document Type = Unknown

queryParameterName	queryParameterDisplayType	queryParameterDataType	phraseGroupID
referenceType	display_as_picklist	STRING	Z31
referenceType_othervalue	display_as_string	STRING	
referenceAuthor	display_as_string	STRING	
year	display_as_integer	STRING	
referenceTitle	display_as_string	STRING	
referenceSource	display_as_string	STRING	
referenceTestLab	display_as_string	STRING	
referenceReportNo	display_as_string	STRING	
referenceLegalEntityId	display_as_string	STRING	
referenceLegalEntityStudyNo	display_as_string	STRING	
referenceReportDate	display_as_date_range	STRING	
remarks	display_as_string	STRING	

Table 0-13 Query Literature Inventory Parameters

The next following table shows the phrase values for Phrase Group ID Z31.

phraseID	phraseText	locale
NOT_SPECIFIED		
1586	study report	en_GB
266	company data	en_GB
1443	publication	en_GB
1486	review article or handbook	en_GB
1542	secondary source	en_GB
811	grey literature	en_GB
1342	other:	en_GB

Table 0-14 Query Literature Inventory Phrase Values

8.1.2. executeLiteratureInventoryQuery

Purpose	Execute the given literature inventory query (see Figure 0-3 Query Execution Definition Structure).

IN	• The open (logged in) session as <tns:Session>
	• The Query Execution Definition which defines
	  o Query Name
	  o Query Parameter List
	  ▪ Query parameter name
	  ▪ Query parameter value

OUT	• A list (<tns:queryResultList>) of Document References containing
	  o Document Type
	  o Last Modification Date
	  o Unique Key
	  o Description
	  o Notes

Exceptions	• InventoryEngineNotAvailableFault Is thrown when e.g. the service is not available to fulfil the request • InventoryEngineFault Is thrown if Inventory Engine is not able to fulfil your request. This may happen if the Engine caught a back-end Exception.

8.1.3. putLiteratureInventoryEntry

Purpose	Stores the given literature inventory entry.
IN	• The open (logged in) session as <tns:Session> • The Literature Inventory Entry as <tns:literatureInventoryEntry>
OUT	• Boolean (direct Boolean xml binding) True to explicitly notify that everything was OK.
Exceptions	• InventoryEngineNotAvailableFault Is thrown when e.g. the service is not available to fulfil the request • InventoryEngineFault Is thrown if Inventory Engine is not able to fulfil your request. This may happen if the Engine caught a back-end Exception.

8.1.4. putLiteratureInventoryEntries

Purpose	Stores the given list of inventory entries. This could be used as bulk upload.
IN	• The open (logged in) session as <tns:Session> • List of Literature Inventory Entries as <tns:literatureInventoryList>
OUT	• Boolean (direct Boolean xml binding) True to explicitly notify that everything was OK.
Exceptions	InventoryEngineNotAvailableFault Is thrown when e.g. the service is not available to fulfil the request InventoryEngineFault Is thrown if Inventory Engine is not able to fulfil your request. This may happen if the Engine caught a back-end Exception.

8.1.5. deleteLiteratureInventoryEntry

Purpose	Deletes the given literature inventory entry
IN	• The open (logged in) session as <tns:Session> • Literature Inventory Entry to delete as <tns:literatureInventoryEntry>
OUT	• Boolean (direct Boolean xml binding) True to explicitly notify that everything was OK.
Exceptions	• InventoryEngineNotAvailableFault Is thrown when e.g. the service is not available to fulfil the request • InventoryEngineFault Is thrown if Inventory Engine is not able to fulfil your request. This may happen if the Engine caught a back-end Exception.

8.1.6. deleteLiteratureInventoryEntries

Purpose	Deletes the given list of literature inventory entries
IN	• The open (logged in) session as <tns:Session> • List of Literature Inventory entries as <tns:literatureInventoryList>
OUT	• Boolean (direct Boolean xml binding) True to explicitly notify that everything was OK.
Exceptions	• InventoryEngineNotAvailableFault Is thrown when e.g. the service is not available to fulfil the request • InventoryEngineFault Is thrown if Inventory Engine is not able to fulfil your request. This may happen if the Engine caught a back-end Exception.

9. Messaging Engine

9.1. Interface Methods

This section describes the operations of the Messaging Engine. The Messaging Engine can be used to show a document on all IUCLID 5 clients where a given user is logged in.

9.1.1. broadcastShowDocument

Purpose	This operation sends a message to all logged in IUCLID 5 clients and requests that clients that are operated by the given user (specified as receiver) open the specified document.
IN	• The open (logged in) session as <tns:Session> • The receiver (specifies the user that receives the message) as <tns:Receiver> The unique key of the document to be shown in the receivers IUCLID 5 client as <tns:documentReferencePK >
OUT	n.a.
Exceptions	• MessagingEngineNotAvailableFault thrown when e.g. the service is not available to fulfil the request • MessagingEngineFault thrown if the Service is not able to fulfil your request. This may happen if the Engine caught a back-end Exception.

10. Reference Tracker Engine

The reference tracker engine tracks the linkage between IUCLID 5 documents. The methods of the Reference Tracker Engine can be used to find out which documents reference a document and which documents are referenced by a document. The Reference Tracker Engine offers a very convenient way to get information on the relations between documents without the need to scan the XML of the documents for references.

10.1. Interface Methods

This section describes the operations of the Reference Tracker Engine. The Reference Tracker Engine can be used to get the inbound and outbound references of a IUCLID 5 document.

10.1.1. queryInbound

Purpose	This operation answers all references from other documents to the given document reference.
IN	• The open (logged in) session as <tns :session> • The unique keys of document for which to query the inbound references as <tns:documentReferencePK >
OUT	• A list of all documents that reference the document given in the IN parameter as list of <tns:documentReference >
Exceptions	• ReferenceTrackerEngineNotAvailableFault thrown when e.g. the service is not available to fulfil the request • ReferenceTrackerEngineFault thrown if the Service is not able to fulfil your request. This may happen if the Engine caught a back-end Exception.

10.1.2. queryOutbound

Purpose	This operation answers all references from the given document reference to other document.
IN	• The open (logged in) session as <tns :session> • The unique keys of document for which to query the outbound references as <tns:documentReferencePK >
OUT	• A list of all documents that are referenced by the document given in the IN parameter as list of <tns:documentReference >
Exceptions	• ReferenceTrackerEngineNotAvailableFault thrown when e.g. the service is not available to fulfil the request • ReferenceTrackerEngineFault thrown if the Service is not able to fulfil your request. This may happen if the Engine caught a back-end Exception.

11. Resource Engine

IUCLID 5 defines several server based resource files (e.g. phrases.xml), which depends on the installed server version of IUCLID 5.

One example for such a resource is the file phrase.xml, which contains the language depending text for the so called Pick List values (see Figure 0-10 Country Phrase List). A Web Service Client also needs the information of the phrase.xml, because the official xml export format only contains the PhraseID. The Web Service Client developer has to create his own Phrase Manager in order to get the language depending text of the PhraseID.

The interface allows Web Service clients to access the phrases.xml via the Web Service and in that case they always getting the correct phrases.xml file of the current IUCLID 5 server. Otherwise they have to manually download the file from the IUCLID web side to getting the phrase content.

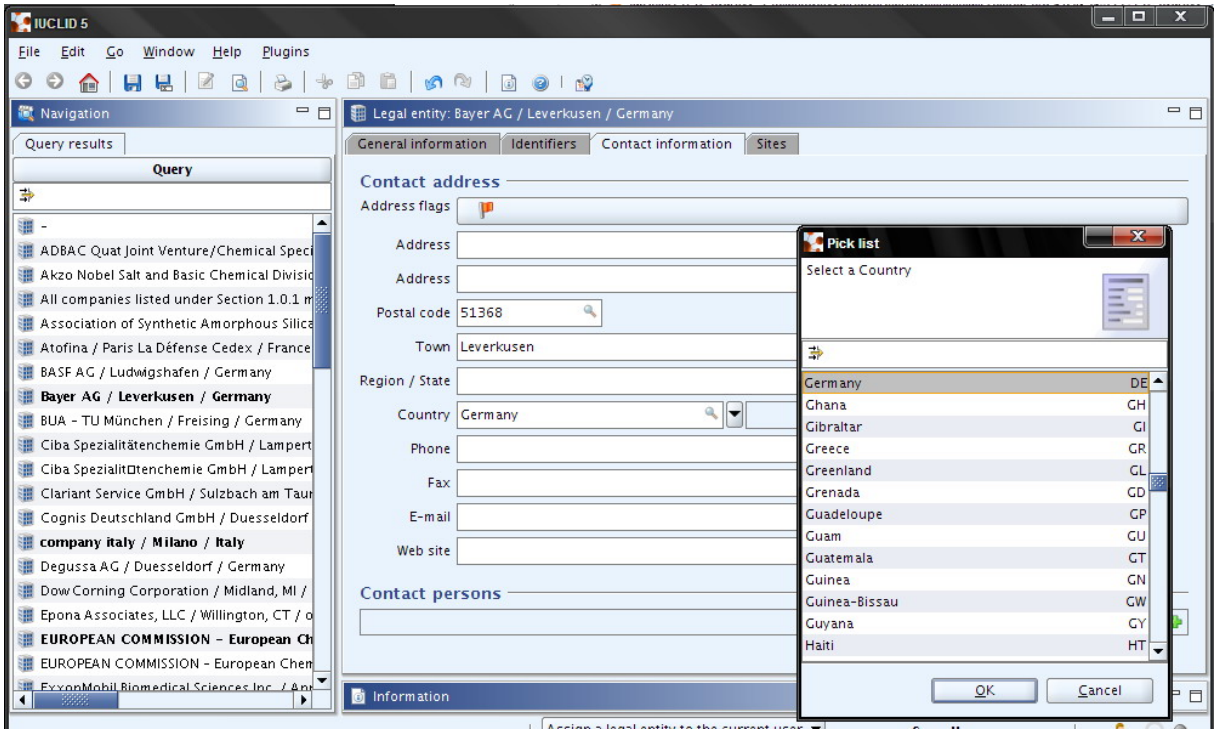

Figure 0-10 Country Phrase List

Besides the phrase content it might also be necessary to load the layout information, which defines the endpoint structures and the definition of the legislation sections.
The following list shows the available resource ID's:

- phrases-xml
- layout-xml
- legislation-sections-xml

11.1. Short Introduction to Data Model

The Resource Engine defines two important data structures:

- Iuclid5Resource
  A iuclid5Resource object contains the requested resource as content and the ResourceDefinition as meta data.
- ResourceDefinition
  A ResourceDefinition object contains the ID of the resource and the optional language.
  - ResourceID
    the resource id specifies the requested resource
    - phrases-xml
    - layout-xml
    - legislation-sections-xml
  - Language
    the language for the requested resource. If you have installed the correct languages, it might be possible to load the phrases.xml for several languages. Please refer to the official IUCLID 5 web side for more information about released languages.

Note:
Please refer to the official IUCLID 5 website for more information about released languages and how to install them.

The following diagram illustrates this structure:

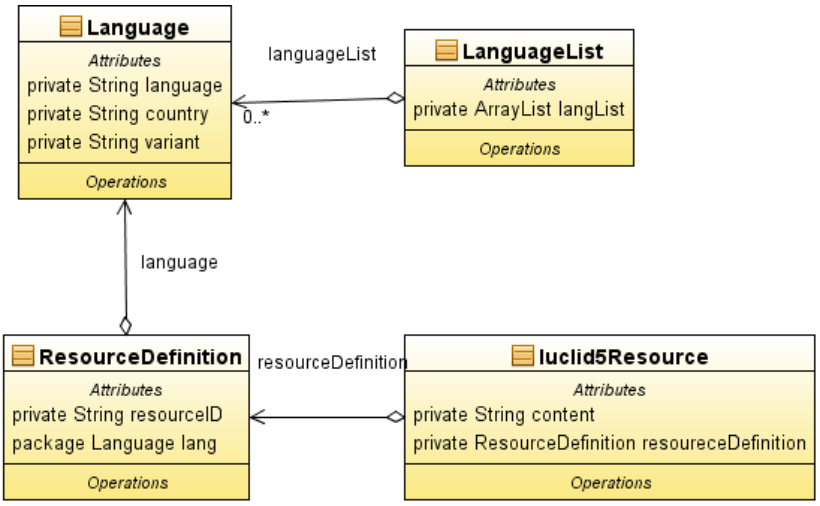

Figure 0-11 Structure of the Resource Engine Object

11.2. Interface Methods

This section describes the operations of the Reference Tracker Engine. The Reference Tracker Engine can be used to get the inbound and outbound references of a IUCLID 5 document.

11.2.1. getAvailableLanguages

Purpose	This operation returns a list of available languages. The languages are installed by using the resource plug-ins at the IUCLID 5 server.
IN	• The open (logged in) session as <tns :session>
OUT	• A list of all available languages <tns:languageList>
Exceptions	• ResourceEngineNotAvailableFault thrown when e.g. the service is not available to fulfil the request • ResourceEngineFault thrown if the Service is not able to fulfil your request. This may happen if the Engine caught a back-end Exception.

11.2.2. getResource

Purpose	This operation returns the requested IUCLID 5 resource as SOAP content.
IN	• The open (logged in) session as <tns :session> • The requested *resource definition* as <tns:resourceDefinition> which contains o resource id o Optional language, if null or empty the web service using the default language associated which the specified user session.
OUT	• The requested resource as <tns:i5resource> (this response is similar to 4.2.4 fetchDocument)
Exceptions	• ResourceEngineNotAvailableFault thrown when e.g. the service is not available to fulfil the request • ResourceEngineFault thrown if the Service is not able to fulfil your request. This may happen if the Engine caught a back-end Exception.

12. Container engine

12.1. Introduction

The current IUCLID 5 Web Service Interface is very powerful as it exposes a very generic set of internal API. This internal API is designed to manage and query all types of documents that pass through the system: substances, reference substances, templates, endpoints, dossiers, etc.

The Document Access Engine is responsible to save the documents at the IUCLID 5 server. Therefore it receives the content as XML, validate it against its official format and then commit it against the server. If you want to implement a dossier import, you have to extract the container document and call the commit message for each included document separately.

As you commit each single document you have to take care of the whole procedure and it is necessary to log each committed document. If one commit fails you have to do the rollback via the web service by yourself.

An additional Web Service has been provided as API for developers to import such i5z-Container using one single call. The problems, like the logging of each containing document are still the same, but each operation is performed at the server side and not at the client side.

12.1.1. Limitations

The transactional dossier upload is subject to some restrictions:

- Asynchronously post status information back to the caller
  (e. g. no progress bar on the status of a single import operation can be displayer, “it is all or nothing”)
- No filtering or processing steps during import that modify the inbound data
- The upload service only guarantee the complete import of a new .i5z dataset (no update guarantee could be foreseen). The IUCLID 5 server, as it is implemented know, do not support transactional operations across multiple documents. Each document (e.g. one Substance, Template or Endpoint Study Record) is treated as a separate object. If you save one of these objects, only one transaction is created for it. If you save the next linked document you will execute this within a new transaction. As IUCLID 5 does not support the transactions across multiple documents we cannot perform a transactional update for one .i5z container. **Dossiers** are always a closed collection of documents. Every document can be identified by the dossier UUID. If a dossier already exists in the system, the import will be rejected. If a problem occurs during the import of a dossier, all documents which have been imported up to this time will be removed by the Web Service. So the integrity on dossier data can be fully ensured. On importing **raw data**, it will be more complex. Before the import starts, it will be verified if the I5Z file contains any substance, category, mixture or template that is already in the database. In this case the import will be fully rejected. The import Web Service for raw data will only add additional documents. No documents will be overwritten (e.g. Reference Substance, Legal Entity). If a problem occurs during the import of a raw data, all already imported documents will be removed by the web-service. So the integrity on raw data can be fully ensured.
- EC Substance Inventory (.i5i) is not supported.

12.1.2. Technical Conditions

The maximum size of a SOAP attachment (dossier .i5z files) can grow very large. In general .i5z files do not exceed 20MB, but exceptions are expected. Due to their internal structure (standard ZIP format) and the expected size they need to be stored to the local file system in order to be decompressed. Where and how temporary .i5z files are stored is an encapsulated implementation detail of the web service respectively the used web service framework.

According the expected size it may take a long time to perform the import, therefore the timeout value of a web service client has to the increased.

12.1.3. Transferring Binary Data

The IUCLID 5 export container is an ordinary ZIP-Container, therefore the API has to provide messages that accept binary data via SOAP.

A SOAP message contains an XML document definition that can be used to exchange structured and typed information between peers in a decentralized, distributed environment. As described above we need to send binary data to the Web Service, but XML may contain only printable characters. In general there are two ways of embedding binary data into an SOAP message.

- *By value* – the binary data are converted into a textual representation, e.g. Base64-Encoding, and than added to the content of an xml element or attribute.
- *By reference* – instead of converting the binary data into a different format, this approach add the data as additional content besides the SOAP message. Within the SOAP-Part of the message you will find an URI-Reference that points the added Attachment-Part. SwA, MIME or DIME are known techniques which are using this approach.

The second approach might be the right choice if you emphasise on performance because you do not have to convert the binary data. But for this approach there is no exiting technology, which could accomplish at the marked. The two technologies SwA (SOAP Messages with Attachments) and DIME (Direct Internet Message Encapsulation) where developed by different industry consortia and are not compatible with each other. If you are using the *By value* approach an ordinary XML-Infoset is used which could be processed by a standard XML parser.

To satisfy the interoperability requirement of web services we used the by value approach to send the binary data.

The following XSD snipped illustrate this message definition.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                                    xmlns:tns="http://echa.europa.eu/schemas/iuclid5/i5webservice/types/"
                                    ... >
  ...
  <!-- INPUT -->
  <xsd:element name="uploadContainer">
        <xsd:complexType>
              <xsd:sequence>
                    <!-- the sender session -->
                    <xsd:element name="session" type="tns:Session"/>
                    <xsd:element name="i5zContainer" type="tns:i5zContainer"/>
              </xsd:sequence>
        </xsd:complexType>
  </xsd:element>
  <!--  OUTPUT -->
  <xsd:element name="uploadContainerResponse">
```

```
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="resultReport" type="tns:resultReport"/>
                </xsd:sequence>
            </xsd:complexType>
    </xsd:element>
    <!--  IUCLID 5 external container -->
    <xsd:complexType name="i5zContainer">
            <xsd:sequence>
                <xsd:element name="dataContent" type="xsd:base64Binary"/>
            </xsd:sequence>
            <xsd:attribute name="fileName" type="xsd:string" use="required"/>
    </xsd:complexType>              ...
</xsd:schema>
```

Code 0-5 MTOM Message Definition

12.2. Container Engine

12.2.1. Short Introduction to Data Model

The following diagram illustrates the data structure of the container engine.

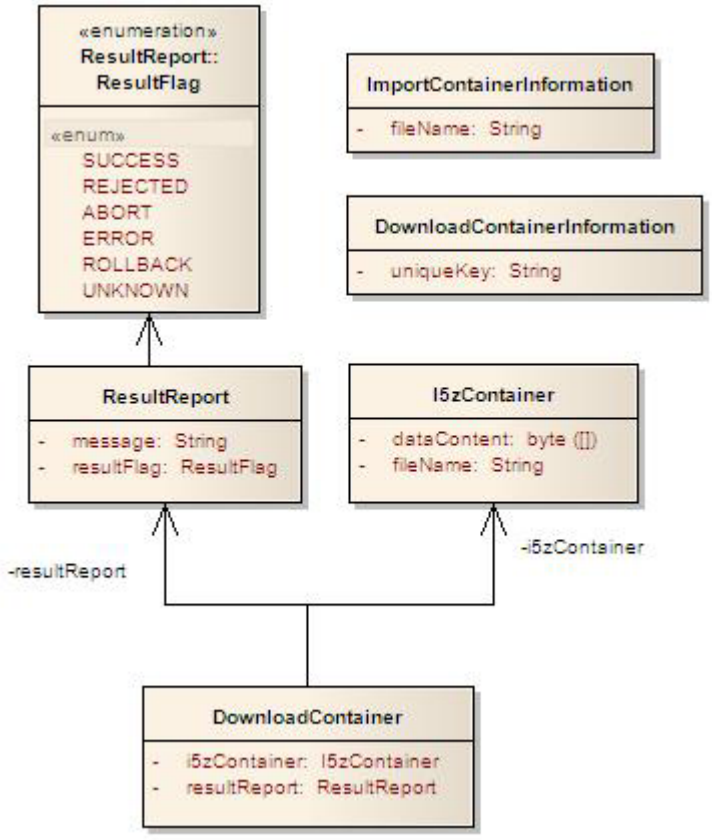

Figure 0-12 Data Structure of Container Engine

12.2.2. Interface Methods

The container engine service is used to upload and download a complete IUCLID 5 export container (*.i5z file).

12.2.2.1. uploadContainer

Purpose	Send a binary IUCLID 5 export container to the web service and starts the transactional import.
IN	• The open (logged in) session as <tns:Session> • The i5zContainer (<tns:i5zContainer tns:fileName="String">…)
OUT	• The import result report (<tns:resultReport tns:resultFlag="SUCCESS">…)
Exceptions	• ContainerEngineNotAvailableFault Is thrown when e.g. the back-end service is not available to fulfil the request • ContainerEngineFault Is thrown if Container Engine is not able to fulfil your request. This may happen if the Engine caught a back-end Exception. • IllegalContentTypeFault This exception is thrown if the received container content type (application/zip) is not allowed. • ContainerContentFault This exception is thrown if the processing of an import container fails.

12.2.2.2. importExternalContainer

Purpose	Call the web service to import an external uploaded i5z-container. The client has to upload the external container file by himself, e.g. via ftp. After the web service is called to importExternalContainer it scans the external upload directory for the specified file name (<tns:importContainerInformation …).
IN	• The open (logged in) session as <tns:Session> • The import container information containing the filename (<tns:importContainerInformation tns:fileName="String"/>)
OUT	• The import result report (<tns:resultReport tns:resultFlag="SUCCESS">…)
Exceptions	• ContainerEngineNotAvailableFault Is thrown when e.g. the back-end service is not available to fulfil the request • ContainerEngineFault Is thrown if Container Engine is not able to fulfil your request. This may happen if the Engine caught a back-end Exception. • IllegalContentTypeFault This exception is thrown if the received container content type (application/zip) is not allowed. • ContainerContentFault This exception is thrown if the processing of an import container fails.

12.2.2.3. downloadContainer

Purpose	Return the
IN	• The open (logged in) session as <tns:Session> • The download container information containing document reference pk oft he requested document. <tns:downloadContainerInformation tns:uniqueKey="-0-0-0-0-0/0"/>)
OUT	• The download container (<tns:downloadContainer>) wich contains the i5z-container (<tns:i5zContainer>) and the export resutl report (<tns:resultReport…).
Exceptions	• ContainerEngineNotAvailableFault Is thrown when e.g. the back-end service is not available to fulfil the request • ContainerEngineFault Is thrown if Container Engine is not able to fulfil your request. This may happen if the Engine caught a back-end Exception. • ExportDocumentFault This exception is thrown if the processing of an export container fails. • OutOfDiskSpaceFault This exception is thrown if the export cache has not enough disk space to write the export container.

12.3. Additional configuration

The following table shows the IUCLID 5 web service configuration options. For more information on how to change the web service configuration, please refer to chapter *1.5 Advanced Deployment* of the Web Service Installation documentation.

Configuration Key	Description
container.engine.external.upload.directory	The external upload directory. Default settings: <java.io.tmpdir>/.containerEngine/externalUpload For apache tomcat this might be: <tomcat-install-dir>/work/Catalina/localhost/ i5wsruntime/.containerEngine/externalUpload
container.engine.**clean**.external.upload.dir	Set to true if the web service should delete the uploaded external file after it was processed. If value is false each uploaded file would remain at the file system. Default value is true.

Table 0-15 Additional Configuration

13. SOAP 1.1 Binding

The IUCLID 5 Web Service using WSDL 1.1 to describe the interface and SOAP 1.2 (released in June 2003) as message format.

There are some frameworks which are used by the industry that do not support SOAP 1.2. In addition a binding for SOAP 1.1 is available.

Nevertheless the default binding is SOAP 1.2 and if someone would like to use SOAP 1.1 he has to download the WSDL files separately. The SOAP 1.1 files are also part of the IUCLID 5 Web Service Runtime application. They can be accessed via the web interface by using the corresponding link (see Figure 0-1 IUCLID 5 Web Service Start Page).

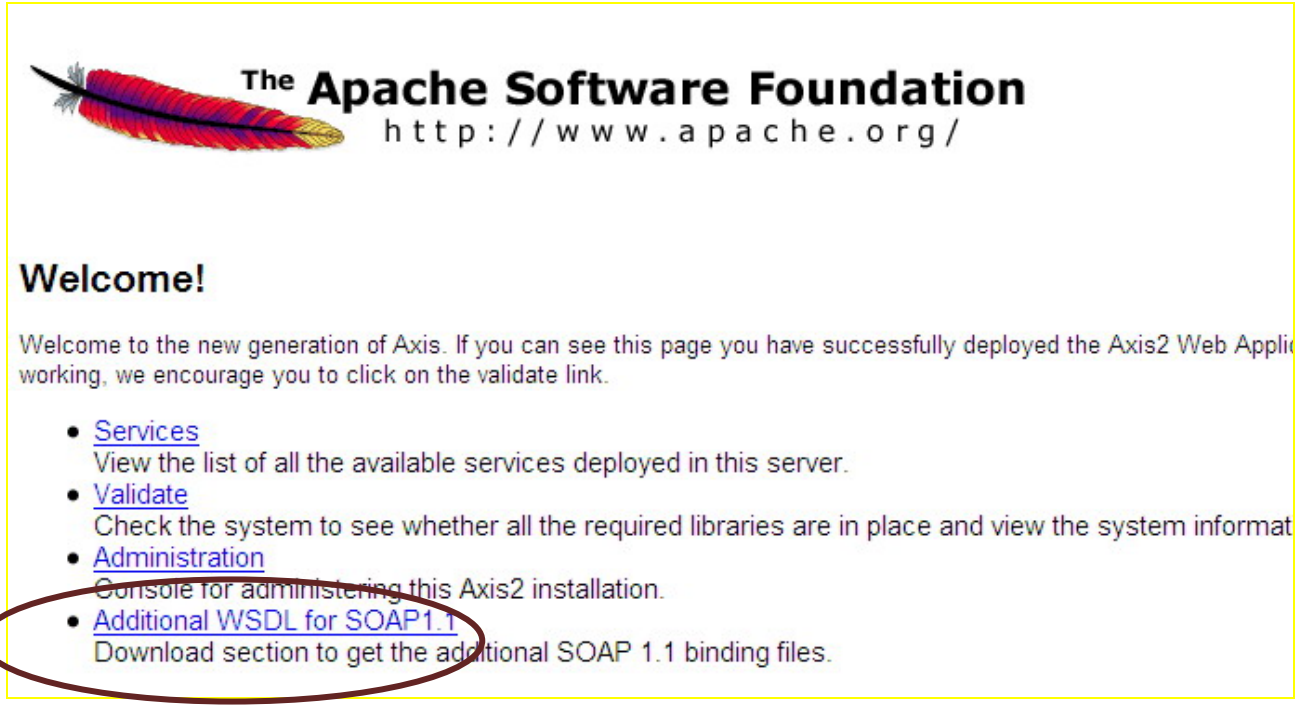
The Apache Software Foundation
http://www.apache.org/
Welcome!
Welcome to the new generation of Axis. If you can see this page you have successfully deployed the Axis2 Web Appli
working, we encourage you to click on the validate link.
Services
View the list of all the available services deployed in this server.
Validate
Check the system to see whether all the required libraries are in place and view the system informat
Administration
Console for administering this Axis2 installation.
Additional WSDL for SOAP1.1
Download section to get the additional SOAP 1.1 binding files.

Figure 0-1 IUCLID 5 Web Service Start Page

After you have clicked at *Additional WSDL for SOAP1.1*, you will enter the SOAP 1.1 download section. This site displays a list of WSDL files which are using SOAP 1.1 binding (see Figure 0-2 SOAP 1.1 Download Section).

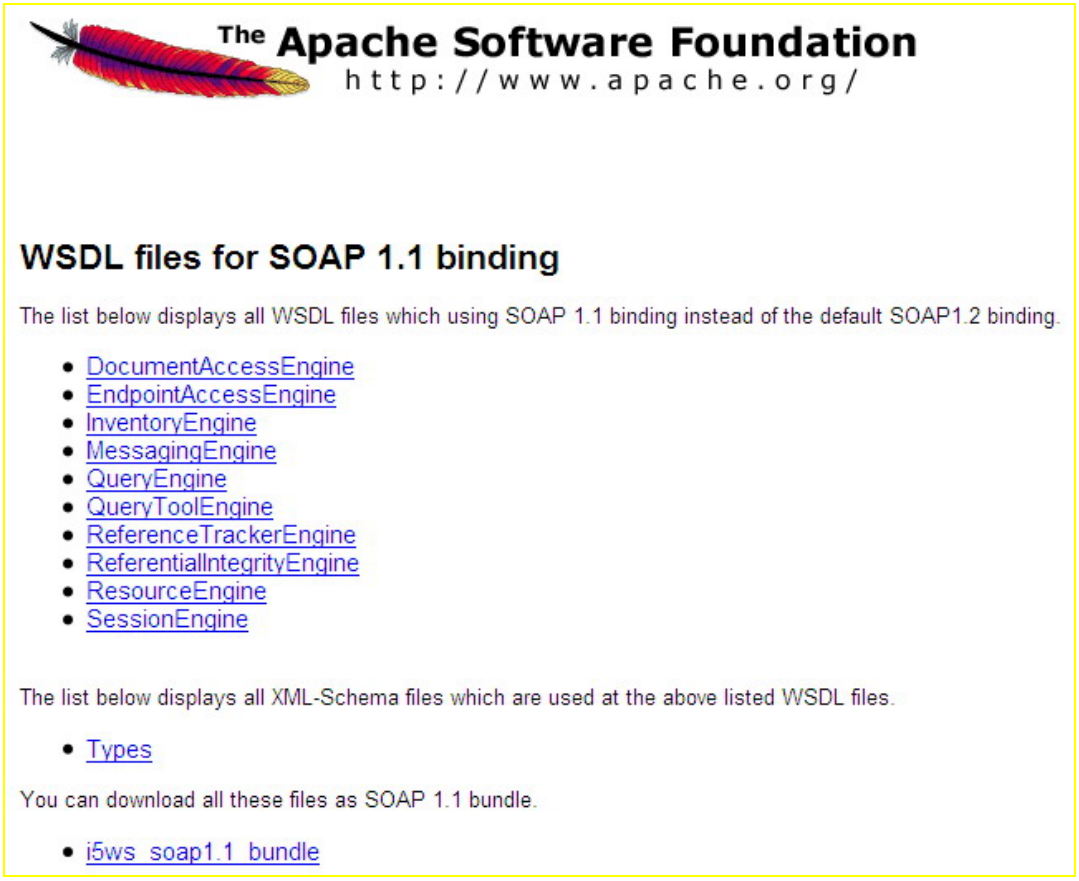

Figure 0-2 SOAP 1.1 Download Section

14. Annex

14.1. Document Identifiers – DocumentReferencePKWs

Each document carries a universally unique identifier (UUID), more precisely this document identifier is a combination of two parts where both consist of a prefix and a UUID.

A document identifier is generated when the document is initially created and remains unchanged for the lifetime of the document. In contrast to earlier releases of IUCLID, IUCLID 5 has been designed as a distributed database system. This means that identifiers to objects in the database must not be unique to the local system only, but they have to be universally unique.

Because the identifier travels unchanged with the document through all the data interchange channels, it must remain unique in whatever location the document gets stored. For instance when a joint submission consortium shares data, the documents in each member's local database share the same identifiers. A great benefit of this approach is the possibility of distributed version tracking.

Schema: [prefix-UUID] / [prefix-UUID]

- The **first** [prefix-UUID] identifies the current document.
- The **second** [prefix-UUID] is either “0” or the so called Dossier Snapshot Key that groups together all documents included in a certain dossier.
- The **prefix** is a four letter acronym. Prefixes in use are for example “IUC4” and “IUC5”. However, the prefixes are not supposed to have any behavioural consequences; they are completely abstract identifiers.

Example document identifier:

- Raw-data: "IUC5-98c376e7-a74c-4913-b818-16ccda4d334d/0"
- Dossier-data: "IUC5-98c376e7-a74c-4913-b818-16ccda4d334d/ IUC5-12345678-abcd-1234-abcd-112233445566"

14.2. Random UUIDs

The standard document identifiers are built from random UUIDs. A UUID is an abstract value with the purpose of being universally unique. More technical background information is available at “RFC 4122: A Universally Unique Identifier (UUID) URN”, (http://www.ietf.org/rfc/rfc4122.txt).

IUCLID 5 uses type 4 UUIDs (pseudo randomly generated). They convey no (traceable) information about who created the identifier. Hence it is impossible to identify the original author of a document by reverse engineering the UUID.

The random UUIDs carry a prefix “IUC5-“ serving as a namespace. It separates the random UUIDs from special UUIDs. The prefix does not and must not be used in any business logic. It is purely abstract.

Pseudo-code to generate a random UUID:

```
/**
 * Factory method for new UUIDs.
 *
 * @return a new generated UUID
 */
public static eu.iuclid.UUID generateNew()
{
   final java.util.UUID javaUUID = java.util.UUID.randomUUID();

   final StringBuilder sb = new StringBuilder( "IUC5" );
   sb.append( '-' ).append( javaUUID.toString() );
   final eu.iuclid.UUID retUUID = new eu.iuclid.UUID( sb.toString() );

   return retUUID;
}
```

Code 0-1 Generate New UUID

14.3. Special UUIDs

For special purposes like migration of data from previous IUCLID versions, a mechanism was needed that conveys the uniqueness of the previous identifiers to the new IUCLID 5 system. Moreover the uniqueness had to be enhanced from the local system to the global IUCLID data realm. The resulting identifiers had to be deterministic and repeatable as multiple migration runs over the same data should not result in duplicates in IUCLID 5 (documents with the same content but different identifiers).

Please note that generating special UUIDs comes with the responsibility to seamlessly interact with the existing mechanisms and systems. Appropriate quality assurance and testing is obligatory and the IMG (IUCLID Management Group at the European Chemicals Agency) should be contacted.

The chosen approach for these special UUIDs is as follows:

- Collect those identifiers from the legacy system that can be used to uniquely identify the concerned object in a global scope. E. g. the IUCLID 5 Migration Tool uses the IUCLID 4 Company-PIN to generate the identifier for a migrated legal entity object. Further immutable properties could also have been considered like the company's central telephone number, VAT/DUNS number, the company's name, etc.
- Optionally chose a different 4-letter prefix (namespace) than IUC5.
  Please note that there is no managing body that controls or assigns prefixes. Collisions (the same prefix used by more than once) cannot be prevented. This does not harm as the prefixes must not be used in any business logic.
- Generate the document identifier

Pseudo code to generate a special UUID:

```
/**
 * Factory method for new special UUIDs from IUCLID4, etc.
 * The method will concatenate string representations of all uniqueness
 * properties and calculate a hashcode (MD5/SHA1/etc) from this long string.
 * Callers of this method MUST commit to the same
 * generation method of these uniqueness properties over the whole data
 * lifetime.
 *
 * Therefore, a legacy system can do something like
 *
 * UUID myUUID = UUID.generateSpecial( "IUC4",
 *                              new String[] { "Company", companyPIN } );
 *
 * or
 *
 * <pre>
 *          UUID myUUID = UUID.generateSpecial( "PRFX",
 *                                      new String[] { "Substance",
 *                                                     substanceType
 *                                                     substanceID } );
 *
 * @param uniqueIDs
 *          array of uniqueness properties
 * @return a new generated UUID
 */
public static UUID generateSpecial(final Object[] uniqueIDs )
{
   final StringBuilder sb = new StringBuilder();
   for( int i = 0; i < uniqueIDs.length; ++i )
   {
      final Object obj = uniqueIDs[ i ];
      sb.append( obj.toString() ).append( '-' );
   }

   final java.util.UUID javaUUID = java.util.UUID.nameUUIDFromBytes(
         sb.toString().getBytes( Utils.UTF8 ) );

   final StringBuilder sb2 = new StringBuilder( "PRFX" );
   sb2.append( '-' ).append( javaUUID.toString() );
   final eu.iuclid.UUID retUUID = new eu.iuclid.UUID( sb2.toString() );

   return retUUID;
}
```

Code 0-2 Create Special UUID

European Chemicals Agency

IUCLID 5 Support

Web Service Technical Manual
May 2009 v 1.0

http://iuclid.eu